**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет інформатики та обчислювальної техніки**

**Обчислювальної техніки**

До захисту допущено:

Завідувач кафедри

Сергій СТІРЕНКО _____

«___»_____20__ р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Комп'ютерні системи та мережі»**

**спеціальності 123 «Комп'ютерна інженерія»**

**на тему: «Модуль корекції помилок для сервісу розпізнавання іменованих сутностей»**

Виконав:
студент IV курсу, групи ІО-61
Харчук Валентин Володимирович _____

Керівник:
Доцент кафедри ОТ, к.т.н.,
Болдак Андрій Олександрович _____

Консультант з нормоконтролю:
Професор кафедри ОТ, д.т.н.,
Сімоненко Валерій Павлович _____

Рецензент:
Доцент кафедри АСОІУ, к.т.н.,
Ліщук Катерина Ігорівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Факультет інформатики та обчислювальної техніки**
**Кафедра обчислювальної техніки**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Сергій СТІРЕНКО_____
«___»_____20__ р.

# ЗАВДАННЯ
## на дипломний проєкт студенту
## Харчука Валентина Володимировича

1. Тема проєкту «Модуль корекції помилок для сервісу розпізнавання іменованих сутностей», керівник проєкту Болдак Андрій Олександрович, к. т. н., доц., затверджені наказом по університету від «07» травня 2020р. №1081-с

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту: технічна документація, теоретичні дані, інтернет-публікації за темою роботи

4. Зміст пояснювальної записки: проведення аналізу предметної області та існуючих передумов для розробки, проєктування і розробка модуля корекції помилок для сервісу розпізнавання іменованих сутностей, проведення тестування розробленого сервісу

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

6. Консультанти розділів проєкту*

* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломного проєкту.

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Нормоконтроль | Сімоненко В.П., проф. | | |

7. Дата видачі завдання _____

Календарний план

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту | Примітка |
|---|---|---|---|
| 1 | Затвердження теми роботи | 01.09.2019 | |
| 2 | Вивчення та аналіз завдання, огляд літератури | 27.12.2019 | |
| 3 | Аналіз існуючих передумов для розробки | 28.01.2020 | |
| 4 | Проектування та розробка сервісу для корекції помилок розпізнавання іменованих сутностей | 10.03.2020 | |
| 5 | Проведення тестування розробленої системи | 20.03.2020 | |
| 6 | Оформлення матеріалів роботи | 20.05.2020 | |
| 7 | Передзахист | 26.05.2020 | |
| 8 | Захист | … | |

Студент Валентин ХАРЧУК

Керівник Андрій БОЛДАК

## Анотація

В даному дипломному проєкті проведено аналіз існуючих рішень у області аналітичного опрацювання природньої мови. Встановлено існуючі передумови для розробки засобу для розпізнавання іменованих сутностей для текстів на українській мові. Спроєктовано і розроблено модуль для вирішення поставленого завдання за допомогою нейронних мереж. Розроблений алгоритм протестовано та результати роботи порівняні із існуючим методом, який базується на ручних правилах. Виявлені переваги та недоліки даного підходу та наведені майбутні дії, які покращать якість роботу модуля.



## Annotation

This Bachelor's work includes the analysis of the existing solutions in the NLP. The existing preconditions for the development of a tool for recognizing named entities for texts in the Ukrainian language have been established. Error correction module for NER module has been designed and developed. The developed algorithm is tested and the results are compared with the existing rule-based method. The advantages and disadvantages of this approach are identified and future actions which will improve the quality of the module are suggested.

# ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

| № з/п | Формат | Позначення | Найменування | Кількість листів | Примітка |
|---|---|---|---|---|---|
| 1 | А4 | | Завдання на дипломний проєкт | 2 | |
| 2 | А4 | ДП 6128. 00.000 ВП | Відомість дипломного проєкту | 1 | |
| 3 | А4 | ДП 6128. 01.000 ТЗ | Технічне завдання | 3 | |
| 4 | А4 | ДП 6128. 02.000 ПЗ | Пояснювальна записка | 59 | |
| 5 | А3 | ДП 6128. 03.000 Д1 | Розподілена система аналітичного опрацювання даних. Схема розгортанн | 1 | |
| 6 | А3 | ДП 6128. 04.000 Д2 | Інтеграція сервісу. Схема взаємодії | 1 | |
| 7 | А3 | ДП 6128. 05.000 Д3 | Схема операцій над вхідними даними. Схема алгоритму | 1 | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

| | | | | ДП 6128. 00.000 ВП | | |
|---|---|---|---|---|---|---|
| | ПІБ | Підп. | Дата | | | |
| Розробн. | Харчук В.В. | | | Відомість дипломного проєкту | Лист | Листів |
| Керівн. | Болдак А.О. | | | | 1 | 1 |
| Консульт. | | | | | КПІ ім. Ігоря Сікорського Каф. ОТ Гр. ІО-61 | |
| Н/контр. | Сімоненко В.П. | | | | | |
| Зав.каф. | Стіренко С. Г. | | | | | |

# ТЕХНІЧНЕ ЗАВДАННЯ

**до дипломного проєкту**
**освітньо-кваліфікаційного рівня__бакалавр___**

на тему: “ Модуль корекції помилок для сервісу розпізнавання іменованих сутностей”

Київ – 2020 року

**ЗМІСТ**

## 1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на аналітичне опрацювання природньої мови.

Область застосування: розпізнавання іменованих сутностей в текстових даних.

## 2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки служить покращення сервісу розпізнавання іменованих сутностей в україномовних текстах.

## 3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка модуля для корекції помилок сервісу розпізнавання іменованих сутностей.

## 4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами для розробки служать науково-технічна література з аналітичного опрацювання текстових даних, застосування алгоритмів машинного навчання та нейронних мереж у вирішення даної задачі, публікації в періодичних виданнях, публікації в Інтернеті щодо даної предметної області.

## 5. ТЕХНІЧНІ ВИМОГИ

### 5.1. Вимоги до розроблюваного продукту

- Ініціалізація великого електронного словника української мови
- Створення та навчання нейронної мережі для розпізнавання іменованих сутностей
- Тестування розробленої нейромережі
- Розробка засобів взаємодії користувача із сервісом.

### 5.2. Вимоги до програмного забезпечення

- Операційна система GNU/Linux
- Python 3.6 і вище
- MongoDB

### 5.3. Вимоги до апаратного забезпечення

- Комп’ютер на базі процесору Intel Core i5 і вище
- Наявність графічного процесору (опціонально)
- Оперативна пам’ять - не менше 8 Гбайт.

# Пояснювальна записка

## до дипломного проєкту

## на тему: «Модуль корекції помилок для сервісу розпізнавання іменованих сутностей»

Київ – 2020 року

**ЗМІСТ**

# ВСТУП

Все, що ми виражаємо (як усно, так і письмово), містить величезну кількість інформації. Тема, на яку ми виражаємось, наш тон, підбір слів, містять додаткову інформацію, яку можна інтерпретувати та отримувати з неї цінність. Теоретично, можна зрозуміти і навіть передбачити поведінку людини, використовуючи таку інформацію.

Дані, отримані з розмов, декларацій, висловлювань людей у соціальних мережах – це все види неструктурованих даних. Неструктуровані дані не можна зберігати у традиційних рядках та стовпчиках таблиць реляційних баз даних, проте це переважна більшість даних, наявних у реальному світі. Такі дані хаотичні, ними важко маніпулювати. Проте, завдяки прогресу в таких дисциплінах, як машинне навчання, відбувається революція щодо цієї теми. Сьогодні мова йде не про спроби інтерпретувати текст на основі його ключових слів, а про розуміння значення цих слів. Таким чином, можна виявити такі мовні звороти, як іронія, або провести аналіз настроїв у тексті.

Даний проєкт присвячений розробці сервісу розпізнавання іменованих сутностей в україномовних текстах. Актуальність роботи підтверджується тим, що досі не існує розроблених інструментів для вирішення даної задачі. Проте, із зростанням даних у вільному доступі та великим досягненням у розпізнаванні сутностей у англомовних текстах, відкриваються всі можливості для розробки відповідного сервісу і для української мови.

У даному проєкті розглянуто класи задач із обробки природньої мови та існуючі рішення. Проаналізувавши всі передумови, спроєктовано власний сервіс для вирішення задачі розпізнавання іменованих сутностей. Розроблена система реалізована у вигляді веб-сервісу, який може бути інтегрований у розподілену систему аналітичного опрацювання даних.

# РОЗДІЛ 1
# ОПИС ХАРАКТЕРУ І АКТУАЛЬНОСТІ ПРОБЛЕМИ

Обробка природньої мови (Natural Language Processing, NLP) – загальний напрям штучного інтелекту та математичної лінгвістики. Даний напрям дає машинам можливість читати, розуміти та отримувати значення з людських мов. Ця дисципліна орієнтована на взаємодію між Data Science та природньою мовою, поширюється у багатьох сферах та галузях. На сьогоднішній день, NLP стрімко розвивається завдяки величезним досягненням у доступі до даних та збільшенню обчислювальної потужності[1].

## 1.1 Класи задач обробки природньої мови

NLP включає широкий круг задач, пов'язаних з обробкою природньої мови (тобто, мови, на якій пишуть і розмовляють люди). Існує набір класичних задач, вирішення яких несе практичну користь.

**Тематичне моделювання** (topic modelling) – це метод виявлення прихованих структур у колекціях текстів чи документів. По суті, це кластеризація текстів та виявлення прихованих тем на основі їх змісту, виявлення та обробка специфічних (ключових) слів і присвоєння їм значень в залежності від їхнього розподілу у тексті. Ця техніка заснована на гіпотезі, що кожен документ складається з набору тем, кожна з яких характеризується набором слів, а це означає, що виявивши ці приховані теми, можна знайти тему всього документу.

Одним з найпоширеніших алгоритмів тематичного моделювання є LDA (Латентне розміщення Діріхле)[10]. Це відносно новий алгоритм (винайдений менше 20 років назад) є прикладом навчання без вчителя (unsupervised learning), який розкриває різні теми, що лежать в основі колекції документів. У методах навчання без вчителя немає вихідної змінної, за допомогою якої можна було б керувати процесом навчання. Більш детальніше, LDA працює наступним чином:

1. Присвоює кожне слово випадковій темі, користувач лише зазначає кількість необхідних тем. Користувачу не потрібно задавати самі теми, а лише вказати число, і алгоритм почне зіставляти всі документи з темами таким чином, щоб всі слова у кожному документі були віднесені до певної уявними теми.
2. Ітераційно проходити кожне слово та переназначати його до кожної теми, беручи до уваги ймовірність, що слово відноситься до цієї теми і ймовірність, що текст згенерований цією темою. Ці ймовірності перераховуються багаторазово, поки відбувається покращення результату.

На відміну від інших методів кластеризації, (наприклад, K-means), які виконують жорстку кластеризацію, тобто, відносять весь текст до однієї теми, LDA присвоює тексту набір тем, які його найімовірніше описують. Наприклад, текст описаний темою 1 на 70%, темою 2 на 20% і темою 3 на 10%.

**Класифікація тексту.** Це випадок, коли необхідно віднести наданий текст до одної із доступних категорій. Один з найпоширеніших з практичної точки зору приклад – класифікація електронних листів на спам або не спам. Другий варіант – багатокласова класифікація, наприклад, віднесення тексту новин до категорії спорт, політика чи погода. Третій варіант застосування текстової класифікації - аналіз тональності. Наприклад, класифікації відгуків користувачів на позитивні, нейтральні та негативні. Оскільки категорій, до яких відносити тексти, можна вигадати багато, текстова класифікація є однією з найпопулярніших практичних задач NLP.

**Розпізнавання іменованих сутностей** (Named Entity Recognition, NER) – це виділення ділянок у тексті, які відповідають встановленому набору сутностей, наприклад, в тексті треба знайти всі локації, персони і організації.

У тексті «Володимир народився у Києві» у рамках даної задачі необхідно виділити, що «Володимир» - це персона, а «Київ» - це локація.

**Вилучення фактів і відношень.** Даний клас задач пов'язаний з розпізнаванням іменованих сутностей. Вилучення відношень із тексту означає розпізнавання зв'язків різних характерів у тексті.

Наприклад, з речення «Дмитро Дубілет – український бізнесмен, IT-директор Приватбанку» персона «Дмитро Дубілет» пов'язаний професійними відношеннями з організацією «Приватбанк». Іншими прикладами є відношення купівлі/продажу, власності, факт народження з додатковою інформацією – датою, місцем та інші.

На перший погляд може здатись, що задача не має очевидного практичного застосування, проте, вона використовується при структуризації неструктурованих даних. Крім цього, це важливий інструмент у пошукових, діалогових системах – коли необхідно проаналізувати питання і правильно зрозуміти, до якого типу він відноситься, а також, які є обмеження на відповідь.

Це, безумовно, не повний перелік задач NLP. По великому рахунку, будь-яку дію, яку можна зробити з текстом на природній мові, можна віднести до задач NLP, просто вищеперелічені задачі зараз найпоширеніші і вже мають очевидні практичні застосування.

### 1.2 Складність вирішення задач обробки природньої мови

Найбільшою складністю у задачах NLP є робота із природньою людською мовою. У будь-якій мові спостерігаються такі явища, як багатозначність (одне слово може функціонувати в кількох різних значеннях, наприклад, коса – знаряддя і волосся) та омонімія (слова, які пишуться однаково, але фонетично відрізняються, або однаково звучать, але різні за написанням). І тому для людини легше зрозуміти, що ключ від замка у дверях – це не одна і та ж сама річ, що і ремонтний ключ. Чи фраза «Press space bar to continue» перекладається на українську мову як «Для продовження натисніть пробіл», а не «Бар космічної преси продовжує працювати».

Іншим прикладом складності у роботі з природньою мовою є анафора. Це явище, коли зрозумілість одного виразу залежить від іншого, що

зустрічався раніше в тексті. Наприклад, у фразі «Мама спекла пиріг. Він був дуже смачним» займенник «Він» відноситься до предмета «пиріг» - це зрозуміло для людини з контексту фрази, але щоб таке зрозумів комп'ютер – треба докласти чимало зусиль. Така задача зараз вирішена з поганою якістю, спроби покращити результат активно продовжуються.

Ще одна складність – це явище еліпсису, тобто спеціальні пропуски у фразах, які мають заповнюватись автоматично за контекстом. Наприклад, у фразі «Іван з'їв червоне яблуко, а Наталка - зелене» для людини легко зрозуміти, що Наталка з'їла зелене яблуко, проте домогтися, щоб машина теж зрозуміла це – непросто. Задача відновлення еліпсису зараз вирішується на малих текстах і має погану якість теж, тому для практичного застосування потрібно значно покращувати результат.

### 1.3 Методи та засоби вирішення задачі розпізнавання іменованих сутностей

Одна з найпопулярніших задач NLP – розпізнавання іменованих сутностей (NER, Named Entity Recognition). Головна ціль NER – виділити спани сутностей в тексті (спан – неперервний фрагмент тексту)[11].

Наприклад, є текст новин, і задача полягає у виділенні сутностей (деякий заздалегідь зафіксований набір – персони, локації, організації, дати і т. д.). Тобто, задача NER – зрозуміти, що «1 січня 2020 року» - це дата, «Стів Джобс» - персона, а «ООН» - організація.

Що саме іменовані сутності? В першій, класичній постановці, яка була сформульована на конференції MUC-6 в 1995 році, - це персони, локації та організації. З тих пір з'явилось декілька доступних корпусів (набір текстів, множина документів), в кожному з яких є свій набір сутностей. Зазвичай до персон, локацій і організацій додають ще нові типи сутностей. Найпоширеніші з них – числові (дати, грошові суми, валюти), а також сутності MISC (від miscellaneous – інші іменовані сутності, наприклад «iPhone X»).

Неважко зрозуміти, що навіть якщо навчитись добре виділяти в тексті сутності, навряд чи це викличе великий інтерес у замовників. Хоча, деяке практичне застосування є і у задачі в класичній постанові.

Один з сценаріїв, коли рішення в класичному варіанті може бути корисним – структуризація неструктурованих даних. Нехай є який-небудь текст (або набір текстів), і дані з нього треба внести до бази даних (таблиці). Сутності з класичного варіанту можуть відповідати рядкам таблиць або заповнювати деякі колонки таблиць. Відповідно, для того щоб правильно заповнити таблицю, треба спочатку виділити в тексті ті дані, які планується внести (зазвичай після цього є ще один етап – ідентифікація сутностей, на якій виділяються однакові сутності, наприклад, коли «ООН» і «Організація Об'єднаних Націй» відносяться до однієї організації, проте, задача ідентифікації або entity linking - це вже інший клас задач).

Є декілька причин, чому NER одна з найпопулярніших задач NLP. По-перше, виділення іменованих сутностей – це крок в сторону розуміння тексту. Це може мати як самостійну цінність, так і допомагати вирішувати інші класи задач NLP. Так, якщо ми знаємо, де в тексті виділені сутності, то можна знайти важливі для інших задач фрагменти тексту. Наприклад, виділити лише ті абзаци, де зустрічаються сутності якогось визначеного типу, а потім працювати лише з ними. Допустимо, надійшов лист, і було б корисно зробити сніпет частини листа, де є щось корисне для користувача, а не просто перший рядок тексту. Якщо вміти виділяти іменовані сутності, то можна залишити в якості сніпету ту частину, де є лише ті сутності, які нас цікавлять. Або в тексті листа виділити кольором потрібні частини (або важливі для нас сутності) для зручності подальшої роботи.

По-друге, сутності – це колокації, їхнє виділення важливе для багатьох задач. Допустимо, дано назву іменованої сутності, і, якою вона б не була, ймовірніше всього, вона неперервна, а, отже, всі дії потрібно проводити над нею як над одним блоком. Наприклад, перекласти сутність з однієї мови на іншу. Тоді треба буде робити переклад всього фрагменту як цілого тексту, а не

пробувати розбити на декілька незалежних фрагментів і перекладати кожен окремо. Вміння визначати колокації корисно і для інших задач – наприклад, для синтаксичного парсингу.

Без NER важко уявити і рішення багатьох інших задач NLP, наприклад, розв'язування займенникових анафор або побудова систем питальння-відповідь. У випадку анафори, легше зрозуміти до якого предмета відноситься займенник. Наприклад, треба проаналізувати текст «Прискакав принц Чармінг на білому коні. Принцеса вибігла назустріч і поцілувала його.» Якщо виділити «принц Чармінг» як сутність виду персона, то значно легше буде навчити машину розуміти, що принцеса поцілувала скоріше всього принца, а не коня.

При побудові систем питання-відповідь виділення іменованих сутностей може допомогти наступним чином. Якщо в пошуковій системі зробити запит «Хто зіграв головну роль в фільмі «Одного разу в… Голлівуді»», то скоріше всього отримаємо правильну відповідь. Це досягається за допомогою NER в тексті запиту: виділяються сутності (фільм, роль і так далі), що допомагає краще зрозуміти, що хочуть у запиті, і далі відбувається пошук у відповідних інформаційних джерелах.

Напевно, найважливішою причиною популярності NER є гнучкість постановки задачі. Іншими словами, можна виділити лише необхідні види сутностей. Можна виділяти будь-які потрібні для нас фрагменти тексту, які відрізняються будь-якими ознаками від іншого тексту. В результаті, можна підібрати індивідуальний набір сутностей для конкретної практичної задачі, розмітити корпус тексту лише ними і навчити модель. Такий сценарій найбільш популярний, і це робить NER одною з найпоширеніших задач в NLP галузі.

**Складності, які виникають в процесі вирішення задачі розпізнавання.** Чому задача NER ще не вирішена повністю і комерційні замовники досі готові платити за неї великі гроші? Здавалось, все просто: зрозуміти, який фрагмент тексту виділити, і потім виділити його. Але в процесі розробки виникають різні складності.

Першою складністю, яка заважає при вирішенні найрізноманітніших задач NLP (включаючи NER) – є різного роду багатозначності, про які вже згадувалось раніше. Але існує окремий вид омонімії, який має безпосереднє відношення до задачі NER – одним і тим же словом можуть називатись сутності різних видів. Наприклад, у нас є слово «Вашингтон». Це може бути як персона, так і локація. Для того, щоб правильно виділити цю ділянку тексту, треба враховувати багато факторів – локальний контекст (те, про що був попередній текст), глобальний контекст (глобальне розуміння). Для людини це легко зрозуміти, але навчити машину досить непросто.

Друга складність – технічна, але не потрібно її недооцінювати. Як би не визначати сутності, ймовірно, виникнуть непрості випадки – коли потрібно вирішувати, що включати в спан сутності, а що ні (зрозуміло, у випадку якщо сутності – це не щось слабо варіативне, типу поштового адреса; такі тривіальні сутності можна було б виділити простішими методами – написати регулярний вираз і не застосовувати ніяке машинне навчання). Наприклад, у тексті «Вас вітає Магазин Професійних Мисливців» слово «Магазин» - це частина назви. Інший приклад – «Вас вітає «Rozetka» - ваш улюблений інтернет-магазин». У даному випадку слово «інтернет-магазин» не потрібно включати в анотацію – це лише опис, а не частина назви. Крім цього, якщо все таки включати, то разом із частинкою «ваш улюблений», а цього робити не хочеться. Ще один приклад – «Вам пише магазин зоотоварів «ЗооСвіт»». Тут вже не зрозуміло, відносити фрагмент «магазин зоотоварів» до назви чи ні. Здається, обоє варіантів можуть бути правильними. Але, важливо зробити вибір і зафіксувати це правило для тих, хто буде розмічати весь текст (адже якщо розмітити неоднозначно, то через розбіжності у розмітці тексту модель буде помилятись).

Таких прикладів помилок можна наводити безліч. Тому, для того, щоб отримати правильну, єдину розмітку, треба створити інструкцію для тих, хто буде розмічати текст. Навіть якщо види сутностей прості, треба узагальнити правила виділення сутностей у вхідному тексті. Кваліфіковані розмітники – це

дорого і повільно. Навіть заплативши гроші, не цілком ймовірно, що можна отримати ідеально розмічений текст, адже якщо інструкція складна, то навіть висококваліфікований розмітник може допустити помилки або щось неправильно зрозуміти. Для корекції помилок можуть перевіряти роботу однієї людини інші розмітники, але це збільшує ціну і час. Проте, уникнути цього процесу не вийде ніяк. Для навчання моделей машинного навчання необхідно мати розмічену навчальну вибірку достатніх розмірів.

Через ці дві причини NER досі не став загальнодоступною галуззю NLP.

**Зведення задачі NER до задачі класифікації.** Не звертаючи уваги на те, що сутності можуть складатись з декількох слів, зазвичай задача NER зводиться до задачі класифікації на рівні токенів, тобто, кожен токен вхідного тексту відноситься до одного з декількох можливих класів. Існує декілька стандартних способів це зробити. Найзагальніший називається BIOES-схемою. Схема заключається у тому, щоб до мітки сутності (наприклад, PER для персон, ORG для організацій) додавати відповідний суфікс, який позначає позицію токена в спані сутності. Більш детально:

- B – від слова beginning – перший токен в спані сутності, яка складається з більше ніж одного слова
- I – від слова inside – додається до токена, який знаходиться всередині спана
- E – від слова ending – останній токен сутності, яка складається з більше ніж одного слова.
- S – single, додається до сутності, яка складається з одного токена.

Таким чином, до кожного класу сутностей додається один з наведених суфіксів. Якщо токен не відноситься ні до якої сутності, він помічається спеціальною міткою, зазвичай O чи OUT.

Для прикладу, візьмемо текст: «Тарас Григорович Шевченко народився у селі Моринці». В даному тексті є одна сутність, яка складається з кількох слів – персона «Тарас Григорович Шевченко» і одна проста (складається з одного слова) сутність – локація «Моринці». Розмічений текст відповідно до

анотації буде мати наступний вигляд (схематично зображена мітка йде за кожний словом у дужках): «Тарас (B-PER) Григорович (I-PER) Шевченко (E-PER) народився (OUT) у (OUT) селі (OUT) Моринці (S-LOC)».

Таким чином, BIOES – це спосіб відображення проекції спанів чи анотацій на рівень токенів. Зрозуміло, що при такій розмітці можна однозначно встановити границі всіх анотацій сутностей. Дійсно, про кожен токен ми знаємо, чи насправді сутність розпочинається чи закінчується на ньому, а тому можемо закінчити анотацію сутності на ньому або розширити її на наступні токени.

Переважна більшість дослідників використовує даний спосіб розмітки (або її варіації з меншою кількість суфіксів – BIO чи BIOE), але у цьому способі теж існують недоліки. Головний з них заключається в тому, що дана схема розмітки не вміє працювати з сутностями, які пересікаються в тексті. Наприклад, «КПІ ім. Ігоря Сікорського» - це клас сутності – організація. Але Ігор Сікорський – це персона, і було б добре задати це у розмітці. Якщо використовувати вищеописану схему розмітки сутностей, то передати ці два факти одночасно не вийде ніколи, і, відповідно, токени «Ігор», «Сікорський» будуть належати або до типу сутність або до типу локація.

Існує ще випадок вкладених сутностей: «Кафедра обчислювальної техніки факультету інформатики і обчислювальної техніки КПІ». В даному тексті можна виділити три вкладені сутності, але, відповідно до наведеного вище способу розмітки, можна виділити три сутності, спани яких не будуть пересікатись, або одну сутність, яка буде складатись з усього фрагменту тексту.

Окрім стандартного способу зведення задачі до класифікації на рівні токенів, існує ще і стандартний формат даних, в якому зручно зберігати розмітку для задачі NER (а також для багатьох інших задач NLP). Він називається CONLL-U. Основна ідея формату в тому, щоб зберігати дані у вигляді таблиці, де кожен рядок відповідає одному токену, а колонки – конкретні типи ознак токена (включаючи нормальну форму слова). Зазвичай

дослідники визначають набір ознак, які включати у таблицю, в залежності від специфіки задачі і способу її вирішення.

**Rule-based підхід.** Дану задачу можна вирішувати без машинного навчання – з допомогою rule-based систем. В найпростішому варіанті – використовуючи регулярні вирази. Даний спосіб здається застарівшим і неефективним, проте якщо чітко визначена і обмежена предметна область, сутності чітко визначені та однозначні, то дану задачу NER можна вирішити за допомогою rule-based методів достатньо швидко і якісно.

Наприклад, якщо необхідно виділити електронні поштові адреси – емейли або числові сутності (дати, грошові суми або номери телефонів), регулярні вирази можуть вирішити задачу швидше і ефективніше, ніж машинне навчання.

Але як тільки з'являються мовні багатозначності різного роду (про які було описано вище), такі прості способи перестають добре працювати. Тому, застосовувати їх можна лише на обмежених задачах і на простих і чітко відокремлюваних від іншого тексту сутностях.

**Нейронні мережі.** У даному підході багато залежить від даних, на яких буде проводитись навчання алгоритму (або нейронної мережі). До появи векторного представлення слів (word embedding) головною ознакою токена був його індекс у словнику. Таким чином, для кожного слова формувався булевий вектор розмірністю словника (кількості всіх унікальних слів у тексті), елементи якого мали значення 0, а на місці індекса даного слова у словнику – було значення 1. Окрім словоформи (індекс слова), в якості додаткових ознак часто використовували належність до частини мови (POS-теги), морфологічні ознаки, префікси, суффікси, наявність спеціальних символів в токені і зовнішній вигляд токена. Зазвичай, дуже важливою ознакою токена є вид його написання, наприклад, перша літера велика, інші маленькі; всі літери маленькі; всі літери великі; нестандартний вигляд – перша маленька, всі інші – великі (наприклад, «iPhone»). Якщо токен має нестандартний вигляд, то можна з більшою ймовірністю сказати, що це якась сутність, але навряд чи це

персона чи локація. Дані набори ознак давали можливість вирішувати задачу NER за допомогою алгоритмів машинного навчання, наприклад, Random Forest Classifier або Gradient Boosting Classifier.

Всі можливі ознаки токена – векторне представлення, символьні ознаки і інші додаткові ознаки формують вектор ознак токена, який не залежить від контексту, тому називається контекстно-незалежним. Він не містить інформацію про сусідів конкретного токена у тексті. Для отримання контекстно-залежних ознак використовувались спочатку статистичні підходи «вікном» - бралось вікно розміром у декілька токенів і всі ознаки рахувались у цьому вікні. Проте, такий спосіб ненадійних, оскільки важлива для аналізу інформація може бути на більшій відстані, ніж розмір вікна. Тому вирішено всі контекстно-залежні ознаки формувати на рівні речень стандартним способом – використовуючи двосторонні рекурентні нейромережі LSTM або GRU. Всі можливі незалежні ознаки подаються на вхід до Bidirectional Recurrent NN (яка складається з одного або кількох шарів). Вихід з даної мережі в і-тий момент для і-того токена буде складати контекстно-залежні ознаки, які будуть містити інформацію як про попередні (за це відповідає пряма RNN), так і про наступні токени у тексті (міститься у відповідному значенні зворотньої RNN). Далі вихід з такої мережі можна використовувати для практично будь-яких задач. У випадку NER, найпростішим способом буде використовувати повнозв’язний softmax шар з кількістю виходів рівній кількості класів сутностей. Таким чином, буде отримано ймовірності належності токена до кожного типу сутності.

Недолік даного методу в тому, що мітка токена визначається незалежно від міток інших токенів. Тобто, контекст враховується, але саме значення мітки сусідніх токенів – ні. Наприклад, незалежно від контексту токенів, мітка I-PER може зустрічатись лише після B-PER або I-PER. Стандартний метод, який враховує сусідні мітки – використання CRF (Conditional Random Fields). Головне перевага – це те, що CRF оптимізує всю послідовність міток у реченні цілком, а не кожний елемент окремо.

**Комбінований спосіб.** Комбінований спосіб або Гібридна NER система – це поєднання методів, заснованих на правилах і машинному навчанні, з використанням сильних сторін кожного. Даний спосіб є досить гнучким, адже його можна використовувати для різноманітних задач, і залежно від типу завдання будувати необхідний алгоритм, що може покращити результат. Попри все, недоліки залишаються такі ж, як і в підході, заснованому на ручних правилах. Такий метод також вразливий до набору сутностей і мовних особливостей.

**NER в українській мові.** Задача NER, як і інші класи задач NLP більш поширені для англійської мови, ніж будь-якої іншої. Існують треновані нейронні моделі, розмічені корпуси для навчання, на платформі Kaggle регулярно проводяться змагання на краще вирішення задачі NLP, але все зазвичай на англійській мові. Найпопулярніші бібліотеки для роботи з NLP, такі як CoreNLP, NLTK, TextBlob, Gensim, SpaCy відображають повний спектр наявних інструментів, проте всі вони засновані на англійській мові, і лише починають розширювати всі свої можливості на інших мовах. Українська мова залишається на другому плані в сфері аналітичної обробки природньої мови.

Причинами цьому є декілька факторів: лише нещодавно з’явився стрімкий прогрес у вирішенні задач NLP і протягом останніх років спостерігається набагато більше активність у даній області, ніж у порівняння з минулим десятиліттям. По-друге, відсутність достатньої кількості даних для розробки та тестування рішень, в основі яких лежать алгоритми машинного навчання або нейронні мережі. Тому, задача NER залишається актуальною та перспективною для української мови, чому і присвячена даний проєкт.

### 1.4 Інформаційні ресурси для української мови

**ВЕСУМ** – великий електронний словник української мови. Спочатку був створений групою ентузіастів з метою перевірки орфографії текстів. Проте згодом стала потреба розуміти зв’язок між лемою і словоформою. Перші версії словника підтримували відмінювання слів, але цього було недостатньо, тому словник розширили, додавши для кожної словоформи теги і леми. Словник

знаходиться у відкритому доступі, тому зараз він є актуальним як ніколи для проєктів, пов’язаних з обробкою української мови[12]. Його можна використовувати в наступних цілях:

1. Орфографічний словник - перевірка написання слів, для цього лише використовуються словоформи
2. Повнотекстовий пошук
3. Лематизація – використовується для вирішення задач NLP, наприклад, побудова word embedding
4. Морфологічний аналіз – завдяки наявності тегів частин мов, можна побудувати складні правила зв'язків слів у реченні

Даний список застосувань можна розширяти, адже, по суті, будь-яка робота з текстом розпочинається з обробки, що неможливо без словника.

**Текстовий корпус для навчання.** Для навчання і тестування моделей у даній роботі використовувався розмічений корпус NER-анотацій, який знаходиться в мережі у відкритому доступі. Корпус створений в рамках проєкту lang-uk, містить 229 україномовних текстів, 217 тисяч токенів та майже 7 тисяч розмічених іменованих сутностей[13]. Наявні сутності наступних типів:

- ПЕРС (Персона) – 4060
- ЛОК (Локація) – 1442
- ОРГ (Організація) – 649
- РІЗН (Інше) – 600

Анотація виконана за наступними правилами. Сутності можуть складатись з одного або кількох слів. Будь-яка сутність має містити хоча б одне слово з великої літери, можливий випадок коли слово написано з помилкою, на іноземній мові або весь текст переведений у нижній регістр. До іменованих сутностей не відносять:

- іменники, які з будь-яких причин починаються з великої літери
- назви місяців, хвороб, сортів рослин і інші назви, які пишуться з маленької літери

- загальні абревіатури звичайних іменників, наприклад, ЗМІ, ВНЗ, але ООН, НАТО – це сутності
- суми, дати і інші числа

Загальні правила анотацій:

- сутності мають бути неперервними
- якщо дві сутності знаходяться поряд, все одно мають бути виділені окремо
- якщо одна сутність пересікається (або є частиною) з іншою, то додатково її виділяти не треба, наприклад, «картина «Ніч на Дніпрі»» - «Ніч на Дніпрі» - сутність типу РІЗН, хоча можна було б виділити, що «Дніпрі» - це також сутність типу ЛОК.

Типи сутностей.

- ПЕРС (персона) – ім’я персони, тобто, людини, персонажа, тварини. Зазвичай складається з одного або кількох власних іменників, які пишуться з великою літери.
- ОРГ (організація) – назва компанії, організації, установи, гурту або іншого об’єднання людей. Даний тип сутності також може включати від одного до кількох слів, частина з яких може писатись з великої літери.
- ЛОК (локація) – географічні назви, назви країн, вулиць, адреси. У випадках, коли назва країни чи іншого географічного об’єкту використовується як об’єкт, сутність все одно відносити до типу локація, наприклад, «Україна багата на таланти» - токен «Україна» є типом ЛОК.
- РІЗН (все інше) – всі інші сутності, які не належать до вищеперелічених типів – назви казок, віршів, пісень та інших творів мистецтва, свят, законів, та інших нормативно-правових документів, веб-сайтів, торгівельних марок та інших.

## ВИСНОВКИ ДО РОЗДІЛУ 1

Протягом останнього десятиліття відбулась революція у системах аналітичного опрацювання природної мови. Це можна пояснити двома причинами: вільний доступ до значних текстових корпусів, які можна використовувати у своїх цілях цілком безкоштовно для статистичної обробки, створення моделей машинного навчання, створення словників векторних представлень слів (word embedding) та ін. Друга причина – розвиток рекурентних нейронних мереж, які здатні працювати з неперервними послідовностями (аудіо файли, текстові дані). Саме завдяки вмінню запам’ятовувати попередні стани, нейронні мережі стали розуміти контекст слів у тексті, що дало значне збільшення якості у роботі з аналітичним опрацюванням природньої мови.

В українській мові майже повністю відсутні будь-які рішення, тому розробка сервісу для вирішення завдань на українській мові, а точніше, розпізнавання іменованих сутностей, актуальна зараз як ніколи. Наявність готових засобів, які полегшують опрацювання природньої мови, як ВЕСУМ (великий електронний словник української мови), а також корпус розмічених україномовних текстів дають всі передумови для розробки. В основі сервісу розпізнавання сутностей знаходиться рекурентна нейронна мережа, яка здатна працювати з реченнями заданої довжини. У випадку, коли слово буде новим, тобто, його не буде в словнику, мережа все одно зможе зробити прогноз належності до одного з можливих типів сутностей на основі контексту всіх слів, які знаходяться поруч. Дана перевага використання нейронних мереж у порівнянні з підходом, заснованим на правилах.

Проте для остаточного прогнозу буде використовуватись комбінована система. Власноруч створені правила, які можна легко контролювати, мають велику гнучкість, але їхня головна проблема – у випадку відсутності тега або словоформи у словнику – неможливість проведення аналізу. Це можна вирішувати за допомогою нейромережі. Саме для цього і буде використовуватись сервіс, розроблений у даному проєкті.

# РОЗДІЛ 2.
# ПРОЄКТУВАННЯ СЕРВІСУ РОЗПІЗНАВАННЯ ІМЕНОВАНИХ СУТНОСТЕЙ

В основі підходу, реалізованого в даному проєкті, знаходиться нейронна мережа, тому для кращого розуміння необхідно розібратись як вона влаштована.

## 2.1 Обгрунтування вибору нейронної мережі

**Перцептрон** – це одиниця нейронної мережі. Винайдено перцептрон у 1950-тих роках науковцем Френком Розенблаттом[14]. Сьогодні вже використовуються інші типи нейронів – сигмоїдні, проте спочатку розберемося у перцептронах. Що ж, як вони працюють? Перцептрон приймає на вхід параметри $x_1, x_2, .., x_n$ і повертає просте число – 0 або 1. Наприклад, нехай буде три вхідні параметри: $x_1, x_2, x_3$. Тоді схематичне зображення нейрона буде мати вигляд, зображений на Рис. 2.1.

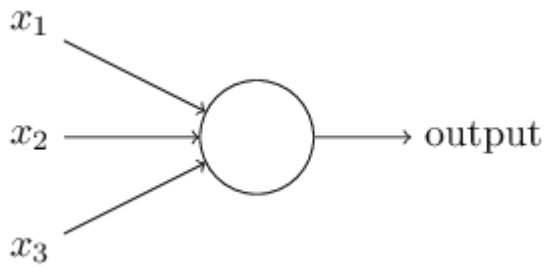



Рис. 2.1. – Схематичне зображення нейрона

Для знаходження результату Розенблатт запропонував наступне просте правило. Він ввів додаткові параметри $w_1, w_2, .., w_n$ – ваги параметрів, тобто, дійсні числа, які характеризують важливість кожного відповідно вхідного параметру. Результат, число 0 або 1, рахується шляхом порівняння зваженої суми вхідних параметрів $\sum_j w_j * x_j$ з певним числом, яке називається пороговим значенням (threshold value). Так само як і ваги, порогове значення – це дійсне число, яке знаходиться в межах [0, 1]. Тому, функцію знаходження вихідного значення нейрона можна представити у наступному вигляді:

$$output = \begin{cases} 0, \text{якщо} \sum_j w_j * x_j \leq threshold \\ 1, \text{якщо} \sum_j w_j * x_j > threshold \end{cases}$$

Саме так працює перцептрон, який являється найпростішою математичною моделлю. Можна уявити, що це пристрій, який приймає рішення на основі зважування доказів. Кожний вхідний параметр – це фактор, який впливає на рішення, так як фактори можуть бути абсолютно різними, у кожного буде своя важливість впливу – вага. А порогове значення необхідно підібрати таким, яке буде влаштовувати достатньо влучно розділення позитивного і негативного рішення (0 і 1).

Очевидно, що перцептрон – не завершена модель людського прийняття рішень. Але зрозуміло, що нейрон може зважувати різні фактори і докази для прийняття рішення. Тому може здатись, що складна нейронна мережа може приймати досить тонкі рішення.

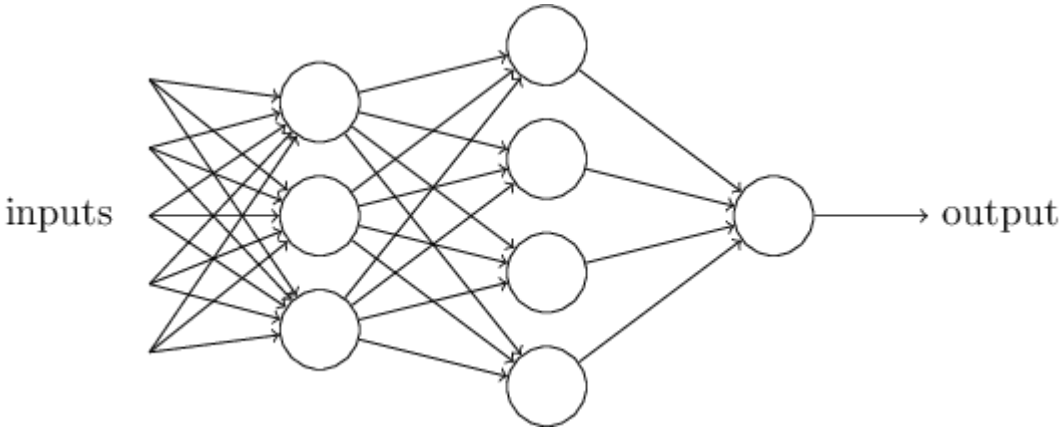



Рис. 2.2. – Схематичне зображення повнозв’язної нейронної мережі

У нейронній мережі, зображеній на Рис. 2.2., у першому шарі нейронів відбувається три простих рішення, завдяки вхідним параметрам (inputs) та відповідним вагам. Але що відбувається у наступному шарі? Кожен з перцептронів у ньому приймає рішення, засноване на зважених результатах першого шару. Таким чином, нейрон у другому шарі може приймати рішення на більш складному і абстрактному рівні, ніж нейрони у попередньому шарі. І, відповідно, нейрони у третьому шарі приймають ще більш комплексні

рішення. Тому, багатошарова нейронна мережа з перцептронів може приймати досить складні рішення. Тепер модифікуємо функцію перцептрона: якщо зробити заміну $threshold = -bias$, отримаємо:

$$output = \begin{cases} 0, \text{якщо} \sum_j w_j * x_j + bias \leq 0 \\ 1, \text{якщо} \sum_j w_j * x_j + bias > 0 \end{cases}$$

Значення bias можна інтерпретувати як значення, необхідне для того, щоб перцептрон повернув 1 (прийняв позитивне рішення). Іншими словами, це міра того, як легко нейрон може вибухнути (активізуватись).

Отже, з'явились параметри $w_j$ і $bias$ кожного нейрона, регулюючи які, можна налаштувати нейронну мережу приймати правильні рішення на вхідних даних. Процес підбору такого набору параметрів, який задовільнить поведінку всієї мережі, називається навчанням.

Проте, існує проблема у навчанні. У випадку, коли нейронна мережа складається з перцептронів, навіть невеликі зміни вагів або bias лише одного нейрона зі всієї мережі можуть призвести до повної зміни вихідного числа з даного нейрона, наприклад, з 0 на 1 або навпаки. Такі зміни можуть призвести до змін поведінки всієї мережі і вона вийде з-під контролю. Цей фактор ускладнює процес поступового змінювання вагів та bias щоб досягти бажаного результату.

Дана проблема вирішується шляхом застосування нейронів іншого виду – нейрони з функцією активації сігмоїда. Такі нейрони подібні до перцептронів, проте модифіковані так, аби невеликі зміни в вагах або bias причиняли невеликі зміни лише у формуванні результату даного нейрона. Це найважливіший факт, який дозволяє нейронній мережі навчатись. А тепер розглянемо такі нейрони детальніше. Схематичне зображення у нього таке ж, як у перцептрона на Рис. 2.1. Вхідні фактори, $x_1, x_2, .., x_n$ тепер можуть приймати значення не тільки 0 або 1, а будь-які дійсні числа у межах [0, 1]. Наприклад, число 0.775 – цілком можливе значення для фактору $x_1$ нейрона з

сигмоїдною функцією активації. Так, як і в перцептрона, у даний нейронів є ваги кожного вхідного параметра - $w_1, w_2, .., w_n$ і $bias$. Але, вихід з такого нейрона не число 0 або 1, а результат функції сигмоїди, яку можна визначити як:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Тому, якщо взяти до уваги всі характеристики нейрона: вхідні параметри $x_1, x_2, .., x_n$, ваги параметрів $w_1, w_2, .., w_n$ і $bias$, отримаємо наступний вираз:

$$output = \frac{1}{1 + \exp\left(-\sum_j w_j * x_j - bias\right)}$$

Для того, щоб краще зрозуміти різницю перцептрона і сигмоїдного нейрона, уявімо, що $z = \sum_j w_j * x_j + bias$ дуже велике додатнє число. Тоді $e^{-z} \approx 0$ і $\sigma(z) \approx 1$. Тому, коли $z$ дуже велике позитивне число, вихід з перцептрона і сигмоїдного нейрона, однакові. Тепер розглянемо випадок, коли $z$ велике від'ємне число. Тоді $e^{-z} \approx \infty$ і $\sigma(z) \approx 0$. Тому, коли $z$ дуже велике негативне число, вихід з обох видів нейронів знову однаковий. Різниця існує лише у тому випадку, коли $z$ лежить у множині чисел, недалеко віддалених від 0. Якщо подивитись на алгебраїчну форму сигмоїди і кусково-заданої функції (яка використовується в перцептроні), то буде зрозуміло, що сигмоїда – це згладжена версія кусково-заданої функції. Саме це забезпечує той факт, що при невеликій зміні вагів або bias вихід нейрона буде змінений теж на невелике число. Можна порахувати цю величину:

$$\Delta output \approx \sum_j \frac{\partial\, output}{\partial\, w_j} * \Delta w_j + \frac{\partial\, output}{\partial\, bias} * \Delta bias,$$

де сума по всіх вагах $w_j$, а $\frac{\partial\, output}{\partial\, w_j}$ і $\frac{\partial\, output}{\partial\, bias}$ означає частинна похідна $output$ по $w_j$ і $bias$ відповідно.

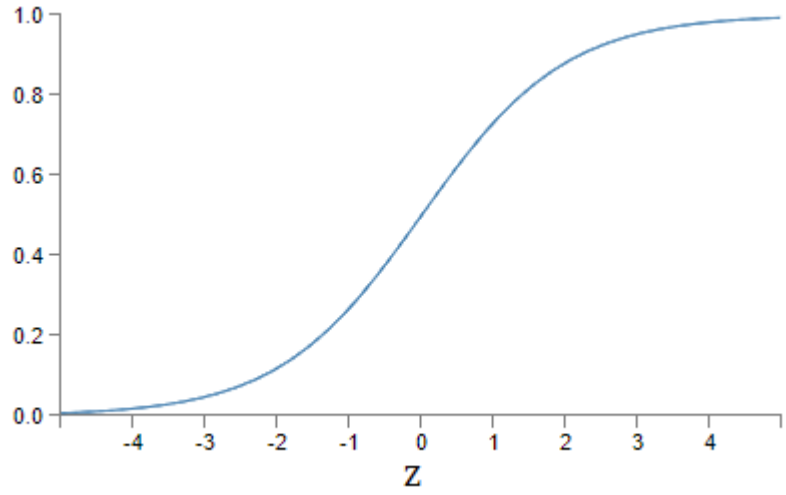



Рис. 2.3 – Функція сигмоїда

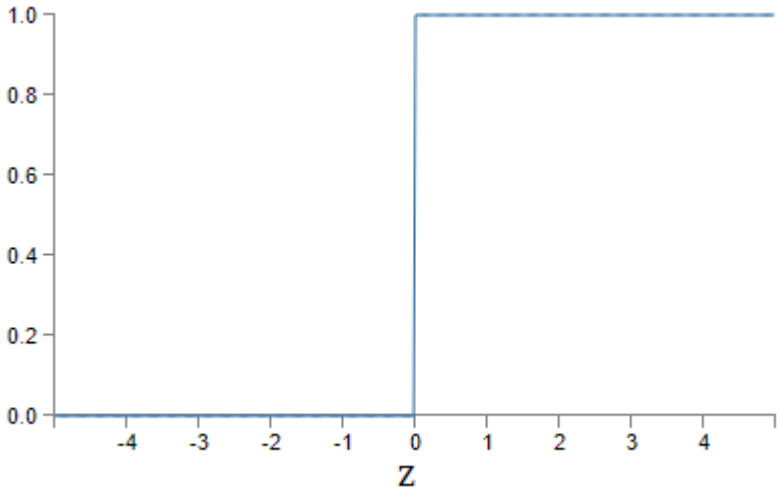



Рис. 2.4 – Кусково-задана функція

Як інтерпретувати вихід з сигмоїдного нейрону? Очевидно, що головна різниця у порівняння з перцептроном - значення може бути не тільки 0 або 1, а будь-яке дійсне число з цього проміжку. Це може бути важливо, наприклад, якщо необхідно використовувати вихідне значення для відображення середньої інтенсивності пікселів вхідного зображення у проміжний шарах мережі. Проте, на останньому шарі, у випадку класифікації необхідно отримати значення 0 або 1, і для цього до виходу з останнього нейрона застосовують порівняння з числом 0.5: менше цього значення – значить вихід 0, більше – 1.

**Навчання нейронної мережі.** Тепер, коли будова нейронної мережі розглянута, виникає питання як навчити її прогнозувати бажаний результат на вхідних даних. Позначимо $y = y(x)$ як результат, до якого має наближуватись передбачення нейронної мережі на вхідних даних $x$.

Для того, щоб розуміти, наскільки якісно модель працює, визначимо функцію помилок (cost або loss function)[15]:

$$C(w,b) = \frac{1}{2*n}\sum_{x}\|y(x) - a\|^2$$

Тут, $w, b$ означають всі ваги і bias у мережі, $n$ – загальна кількість навчальних прикладів, $a$ – передбачення мережі по кожному об'єкту з вхідної вибірки, $\|\upsilon\|$ - це всього лиш позначення довжини вектора. Тому, можна сказати, що $C(w,b)$ квадратична функція, також можна зустріти середньоквадратична функція помилок (mean squared error - MSE). З цього випливає, що $C(w,b)$ завжди додатня, а коли її значення стає близьким нуля $C(w,b) \approx 0$, то передбачення мережі максимально наближене до реальних значень для кожного прикладу в навчальній вибірці $x$. В такому випадку можна сказати, що навчальний алгоритм добре виконався і зміг підібрати такі параметри $w, b$, що помилка близька нуля. В іншому ж випадку, коли значення $C(w,b)$ велике – це означає, що $a$ (прогноз моделі) сильно відрізняється від справжніх значень на великій кількості об'єктів.

Тому, метою навчального алгоритму є мінімізація функції помилок $C$, як функції, яка залежить від параметрів мережі. Необхідно підібрати такий набір параметрів вагів і bias. Це буде зроблено за допомогою градієнтного спуску (gradient descent). У якості функції помилок можна вибирати і інші варіанти, проте середньоквадратична дає найкраще розуміння основ навчання нейронних мереж, тому залишаємось на даному варіанті.

Оскільки головною ціллю алгоритму є підбір параметрів $w, b$ для кожного нейрона, відкинемо зараз всі нюанси (функцію активації, архітектуру мережі, кількість шарів, з'єднання нейронів і тому подібне) і зосередимось на мінімізації параметрів функції помилок. Тому, уявімо, що необхідно

мінімізувати функцію $C(u), u = u_1, u_2, \ldots, u_n$. Для спрощення візьмемо лише два параметри - $u_1, u_2$. В такому вигляді функція помилок буде виглядати наступним чином

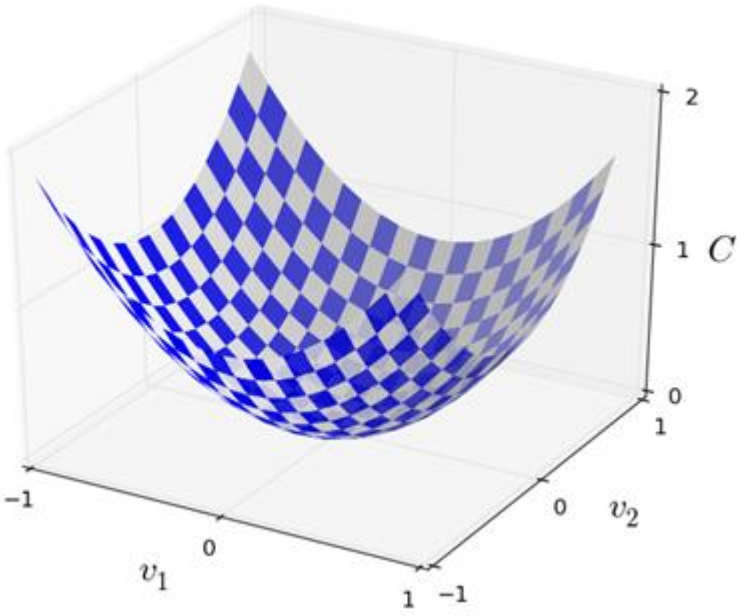



Рис. 2.5. – Функція помилок від двох параметрів

Для вирішення завдання необхідно знайти глобальний мінімум. У випадку, зображеному на Рис. 2.5. можна це зробити візуально, проте це навчальний приклад, у реальних задачах зазвичай функція залежить від багатьох параметрів і набагато складніша, тому візуально визначити мінімум так не вдасться. Як варіант, можна аналітично порахувати мінімум. Для цього необхідно порахувати похідні і потім використати їх для знаходження екстремумів. Це може спрацювати, якщо функція від одного чи кількох параметрів. Але при збільшенні змінних такі розрахунки затрудняться. Тому даний варіант теж не підходить.

Повернемось до зображення функції, вона нагадує долину. Уявимо, як куля поступово скотиться в найнижчу точку. Така ідея – поступового спуску в точку мінімума лежить в основі алгоритму. Похідні розкажуть локальну форму «долини», тобто, в якому напрямку необхідно кулі рухатись. Давайте порахуємо, як зміниться $C$, якщо змістити $u_1$ на $\Delta u_1$і $u_2$ на $\Delta u_2$: $\Delta C \approx \frac{\partial C}{\partial u_1} * \Delta u_1 + \frac{\partial C}{\partial u_2} * \Delta u_2$. В результаті, треба підібрати $\Delta u_1$і $\Delta u_2$ щоб зробити $\Delta C$

від’ємним, тобто, кулька зміститься вниз долини на деякий крок. Вектор $\Delta u$ позначатиме вектор коефіцієнтів кроку спуску: $\Delta u = (\Delta u_1, \Delta u_2)^T$. Визначимо також градієнт функції $C$ як вектор часткових похідних по кожній змінній: $\nabla C = (\frac{\partial C}{\partial u_2}, \frac{\partial C}{\partial u_2})^T$. Перепишемо рівняння для знаходження $\Delta C$:

$$\Delta C \approx \nabla C * \Delta u$$

Тепер, нехай $\Delta u = -\eta * \nabla C$, де $\eta$ - маленьке додатнє число, відоме як крок навчання (learning rate). З цього рівняння виходить:

$$\Delta C \approx -\eta * \|\nabla C\|^2$$

Квадрат у рівнянні завжди забезпечить додатнє число, і тому $\Delta C$ буде від’ємним. Так ми зможемо визначити $\Delta u$ і перемістити положення кульки:

$$u \to u` = u - \eta * \nabla C$$

Застосовуючи дане рівняння ітераційно, ми поступово приблизимось до глобального мінімума. Підсумовуючи, градієнтний спуск – це ітераційний алгоритм, який заключається в знаходженні градієнту $\Delta C$ і потім «рахуючись» в протилежну сторону, «спускатись» до точки мінімума. Схематичне зображення на Рис. 2.6.

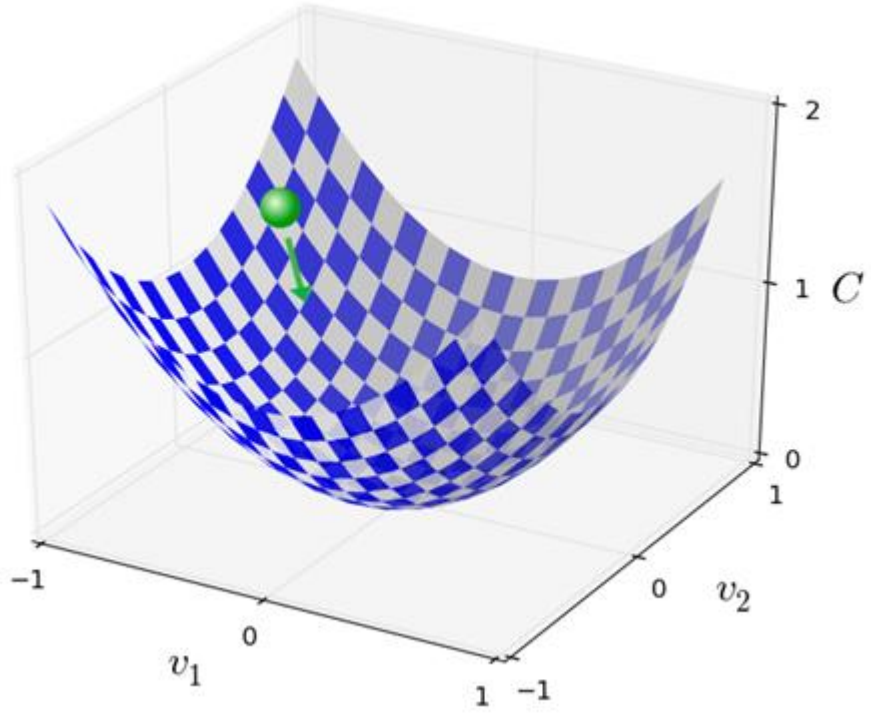



Рис. 2.6. – Мінімізація функції помилок

**Рекурентні нейронні мережі.** У нейронній мережі, яка була розглянута вище, вхідний шар повністю визначав активації нейронів у наступних шарах. Але уявімо, щоб активація залежала не лише від попередніх шарів, а і від активацій в попередні моменти. Наприклад, активація нейрона може залежати від його активацій в попередні рази. А може, залежати не від поточного вхідного об’єкта, а і від попередніх.

Нейронні мережі з такою поведінкою, визначеною залежністю від попередніх часових станів, називаються рекурентними (recurrent neural network). Головна ідея рекурентних мереж полягає в понятті про динамічні зміни в часі[16]. А тому, вони показують хороші результати в роботі з даними, які змінюються з часом. Наприклад, з текстовими чи аудіо даними.

Рекурентні мережі мають багато спільного з традиційними, повнозв’язними: вони також навчаються за допомогою алгоритму градієнтного спуску та методу зворотного поширення помилки (backpropagation). Багато інших ідей, такі як регуляризація, функція помилок, також використовуються у рекурентних мережах.

Для перших моделей справжнім викликом став процес навчання. Причина була у проблемі нестабільного градієнту. Це проблема заключається в тому, що градієнт стає меншим і меншим при поширенні через шари мережі. Як наслідок, навчання на перших шарах стає дуже повільним. Така проблема ще більше проявляється в рекурентних нейронних мережах, оскільки градієнт поширюється не лише через шари, а ще й у часі.

Проте, винайдені нові види нейронів, які називаються довгою короткотривалою пам’яттю (LSTM, Long Short-Term Memory Unit)[17], з ціллю подолання даної проблеми, змогли отримати хороші результати в навчанні моделей. Ключовий компонент такого нейрону – це стан (cell state). Саме завдяки цьому з’єднані нейрони можуть обмінюватись інформацією про попередні стани. Проте, інформацію про стан можна видаляти, цей процес регулюється структурами, яка називаються фільтрами (gates). Фільтри дозволяють пропускати інформацію на підставі значень, отриманих з нейрону

з сигмоїдою. Значення, близьке нуля, означає не пропускати нічого, а близьке одиниці – пропустити всю інформацію.

В LSTM таких три фільтри, які дозволяються контролювати і захищати стан. Перший крок – вирішити, яка нова інформація буде зберігатись в стані нейрона. Це відбувається в 2 кроки: спочатку сигмоїдальний нейрон, який ще називають «шар вхідного фільтра» (input layer gate) вирішує, які значення треба обновити. Потім tanh-шар будує вектор нових значень-кандидатів, які можна додати в стан. Далі треба вирішити, яка інформація повертатись на виході з нейрона. Вихідні дані будуть базуватись на значенні стана, але із застосуванням фільтра. Спочатку застосовується сигмоїдний шар, щоб вирішити, яку саме інформацію з стану необхідно передати. Потім значення стану проходять через tanh-шар, щоб отримати на виході значення в діапазоні від -1 до 1 і перемножити на вихідні значення з сигмоїдного шару, що дозволить вивести лише необхідну інформацію, і цим самим, передати її до наступної LSTM-комірки.

Так виглядає звичайний нейрон LSTM, проте існує багато варіацій. Однією з найпопулярніших модифікацій є додавання фільтрів для додаткових спостережень (peephole connections). З їх допомогою, шари фільтрів можуть бачити стан комірки. Інша модифікація включає об’єднані фільтри забування і вхідні фільтри. В цьому випадку, рішення, яку інформацію потрібно забути, а яку запам’ятати, приймається не окремо, а разом. Ідея полягає в тому, що будь-яку інформацію можна забути лише тоді, коли потрібно на її місце записати щось нове або додавати нову інформацію в стан комірки лише після того, як звільниться місце від старої.

Дещо відрізняються від стандартних LSTM керуємі рекурентні нейрони (Gated Recurrent Units, GRU)[18]. В них фільтри забування і вхідні фільтри об’єднуються в один фільтр обновлення (update gate). Крім того, стан комірки об’єднують з прихованим станом. Крім цього, є ще і інші дрібні зміни. Побудована мережа за допомогою GRU в результаті простіша, ніж стандартна LSTM, а тому її популярність зростає.

## 2.2 Проєктування архітектури мережі для корекції розпізнавання

Як вже згадувалось вище, в даному проєкті основне завдання – виділення в україномовному тексті іменованих сутностей, що буде виконувати нейронна мережа. Виходячи з того, що текст – це послідовність контекстно-пов'язаних слів, тому необхідно рекурентну мережу з типами нейронів LSTM.

Ще одним важливим рішенням є використання векторних представлень слів у реченні – word embedding. Такий підхід зможе не просто перетворити слово у індекс словника, а надасть певного значення. Незважаючи на те, що це буде представлення контекстно-незалежні властивостей, все одно це дасть певного облегшення в навчанні моделі. В процесі навчання можна спробувати різні навчені словники представлень: word2vec, glove. А також спробувати навчити свій в ході навчання моделі. Цей параметр легко контролювати, просто встановити опцію use_pretrained_weights у відповідному шарі нейронної мережі.

Далі, перетворені слова з вхідного речення передаватимуться на вхід до шару LSTM, який буде складатись з двох послідовностей нейронів, з'єднаних у порядку (Forward LSTM) та зворотньому порядку (Backward LSTM). Такий підхід можна назвати одним шаром – двонаправлений (bidirectional LSTM). Виходи з нього передаються на вхід до останнього шару мережі – Output Layer. Завдяки функції активації softmax, на виході мережа буде давати ймовірності належності кожного вхідного слова речення до кожного існуючого типу іменованої сутності, і, відповідно, сутність з найбільшою ймовірністю буде прогнозом моделі. На Рис.2.7. зображена архітектура описаної рекурентної нейронної мережі.

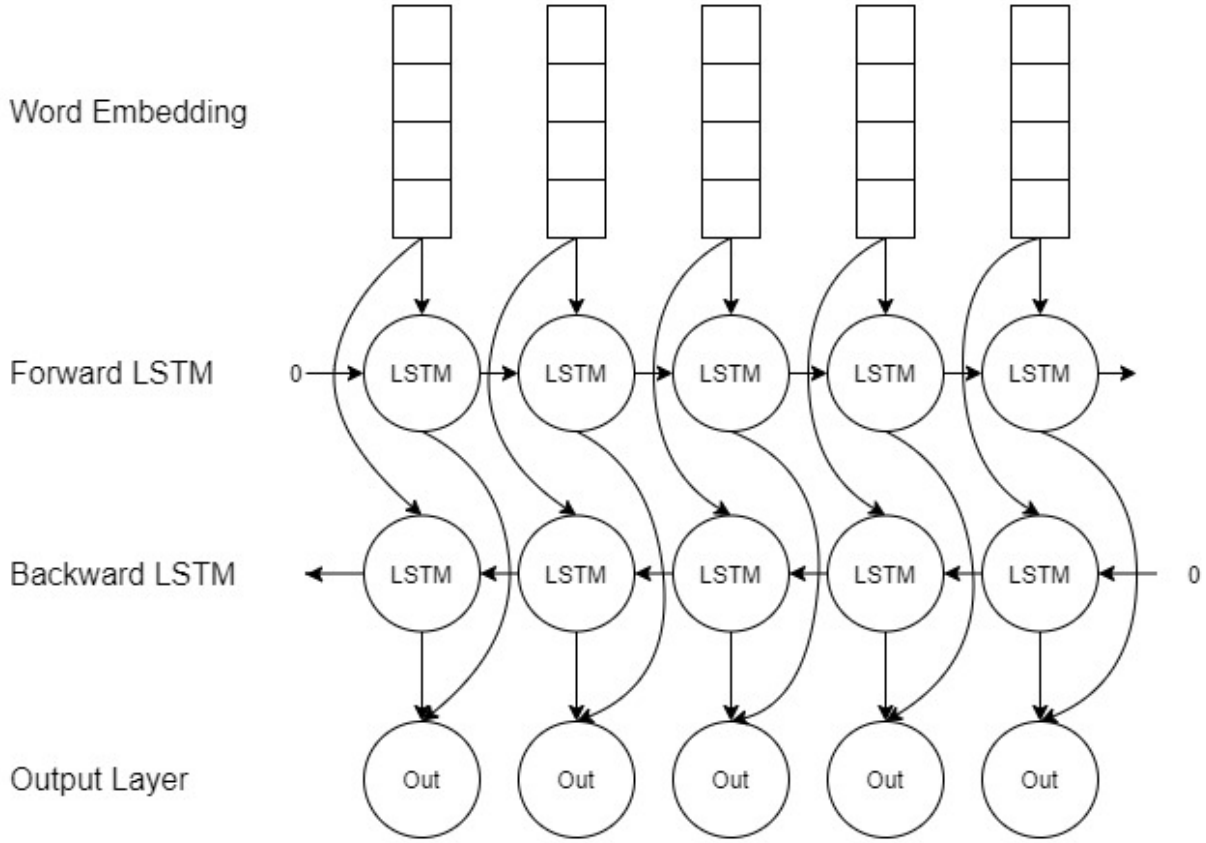



Рис. 2.7. – Архітектура нейронної мережі

**Схеми використання розробленого сервісу.** Розроблений продукт буде функціонувати у вигляді веб-сервісу. Для реалізації використався фреймворк для розробки веб-застосунків - Flask. Перевагою є простий початок роботи та можливість масштабування у майбутньому. Flask не вимагає ніяких архітектур чи будь-яких інших залежностей. Всі перелічені переваги даного інструмента і стали вирішальними у його виборі. За допомогою Flask створено RESTfull API. Використовуватись буде у комбінації з HTTP-протоколом. Гнучність REST дозволяє працювати з будь-якими форматами даних, не обмежуючись лише XML. Тому для вирішення завдання взаємодії клієнта з сервісом дане рішення цілком задовільняє всі потреби. На Рис. 2.8. зображено сценарій використання: користувач відправляє запит із необхідним текстом, а у відповідь отримує список знайдених сутностей з необхідними атрибутами.

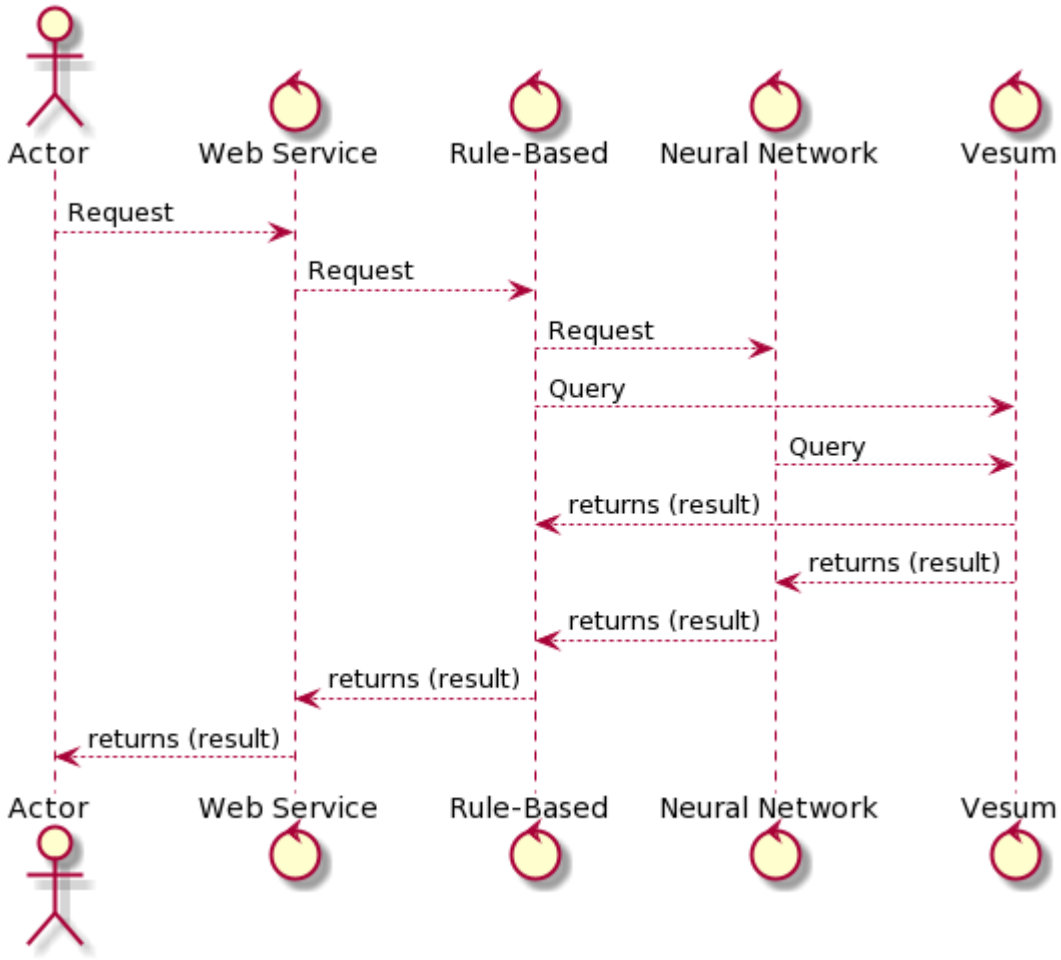



Рис. 2.8. – Схема взаємодії користувача з сервісом

Також розроблений сервіс може бути інтегрований в розподілену систему аналітичного опрацювання даних, схема розгортання наведена в додатку А схемі розгортання, а в додатку А схемі взаємодії зображено схему інтеграції сервісу у вигляді віддаленого сервісу, доступ до якого мають іншу вузли системи.

**Розробка алгоритму попереднього опрацювання даних.** При надходженні нових вхідних даних над ними відбувається ряд маніпуляцій, перед тим як нейронна мережа візьме їх у роботу. І після отримання прогнозів мережі необхідно правильно інтерпретувати результат у зрозумілий для відправника вигляд. Розглянемо всі операції над даними детальніше.

Перш за все, вхідні дані надходять у вигляді тексту (текстового документу). Їх необхідно розділити на речення – виконати токенізацію по реченнях. Далі у кожному реченні виділити слова – токенізація по словах. Після цих двох перетворень необхідно отримати двовимірний масив, де

внутрішні масиви – це набори слів кожного речення, а зовнішній масив – це речення вхідного документа.

Наступний етап заключається у взаємодії з ВЕСУМ. Електронний словник реалізовано у вигляді бази даних NoSQL – MongoDB. Так як в електронному словнику для кожного слова існує три атрибута: поточне слово, його нормальна форма і теги даного слова, для реалізації словника у вигляді бази даних достатньо однієї таблички з трьома колонками. MongoDB обрана через свою простоту у початку роботи і підтримці. Також внаслідок наявної індексації робити пошук по записах таблиці можна буде ще проводити ще швидше. При старті роботи сервісу відбувається перевірка наповненості бази даних, і у випадку відсутності записів викликається процес ініціалізації – зчитується текстовий файл словника і заносяться записи до відповідної таблиці.

Словник потрібний для зменшення кількості унікальних слів у текстах, і, як наслідок, полегшення навчання нейронної мережі. Дана операція відбувається за наступною логікою. Відбувається пошук по кожному слові вхідного документа, виділяється нормальна форма слова. Якщо для всіх знайдених слів нормальна форма однакова – слово заміняється на його нормальну форму, проте якщо знайшлось кілька різних нормальних форм – слово залишається в початковому вигляді (випадок неоднозначності).

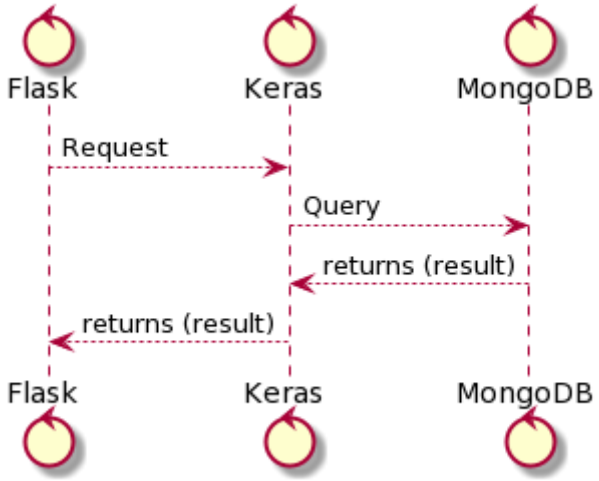



Рис. 2.9. – Діаграма використаних технологій в модулі корекції

Наступний етап – це вирівнювання кількості слів у кожному реченні до одного конкретного числа, в даному розв’язку це 70 токенів. Така операція називається «padding». У випадку, коли кількість слів у реченні більше необхідної, то відкинуться останні слова. Коли ж слів недостатньо, то в кінець речення додаються спеціальні слова типу «PADDING». Це вимушена дія з всіма вхідними реченнями, адже нейронна мережа працює із послідовностями однакових розмірів. Далі необхідно зробити заміну слів на їхні індекси з словника всіх слів. Даний словник сформувався при навчанні нейронної мережі. Для того, щоб нейромережа правильно зрозуміла вхідні слова – вони заміняються на відповідні індекси, вже відомі для неї з процесу навчання. У такому вигляді вхідні дані подаються на вхід до навченої нейронної мережі.

На виході з нейронної мережі отримується для кожного вхідного слова масив ймовірностей належності до кожного наявного тега. Наприклад, якщо модель прогнозує один з трьох класів, то на виході можна отримати список значень (0.25, 0.15, 0.6). Далі, знаходиться індекс із найбільшою ймовірністю, в даному випадку – це 2 (починаючи з нуля). Проте, індекси результуючих класів користувачу теж нічого не повідомляють, тому завдяки словнику тегів перетворюємо індекси на мітки, у нашому випадку це може бути, наприклад, Персона. Така операція проводиться над кожним словом кожного речення, в результаті якої отримуються пари (слово, розпізнана іменована сутність). Далі знайдені сутності об’єднуються, для них визнаються індекс початку і кінця, створюється список всіх знайдених, який і повертається у відповідь для користувача. Весь процес операцій і перетворень над вхідним текстом зображено в додатку А схема алгоритму.

**Навчання нейронної мережі.** У попередньому пункті було показано послідовність всіх етапів обробки та перетворень даних, які відбуваються в процесі роботи сервісу, проте там використовується вже готова, навчена нейронна мережа. Тут детальніше поясниться, на яких даних відбувався процес навчання.

Перш за все, для навчання нейронної мережі необхідні розмічені дані відповідно до єдиної інструкції анотацій. Було використано текстовий корпус, який знаходяться у відкритому доступі в мережі Інтернет, створений в рамках проєкту lang-uk, описаний у першому розділі. Перед безпосереднім навчанням мережі необхідно було зробити деяку обробку і трансформацію даних у необхідний вигляд. У джерелі на кожний документ було дві копії: із розширенням .txt текстовий документ, а із розширенням .ann – файл з анотаціями наявних сутностей у відповідному документі.

Такий формат не підходив, тому було написано функцію, в результаті якої отримано корпуси з розбитими текстами на речення, кожне речення на пари слів і відповідних тегів, у випадку якщо, слово не означає ніякої сутності – йому записується тег «O» (Other). Так було отримано розмічену навчальну вибірку, за допомогою якої можна навчити нейронну мережу і перевірити її якість на відкладеній частині даних. Проте, у такому вигляді модель не вміє працювати з даними.

Тому, як і в попередньому пункті, перед передачею даних до мережі, над ними треба провести декілька додаткових операцій. Спочатку за допомогою ВЕСУМ зменшення кількості унікальних слів. Логіка така ж сама – заміна слова на його нормальну форму, якщо вона однакова для всіх знайдених записів. Далі необхідно вирівняти довжину всіх речень на однакову кількість слів. У наданих текстах встановлено число 70. Тобто, якщо у реченні більше слів – то залишаться лише перші 70 слів, а якщо їх менше, то в кінець речення буде додано спеціальні слова, наприклад, «PADDING». Таке перетворення необхідно для нейронної мережі, адже кожне речення має бути фіксованої довжини, бо в кожному шарі кількість нейронів незмінна і очікує однаковий розмір кожного вхідного текстового прикладу. І остання операція над текстовим корпусом – це заміна слів на їхні індекси з словника всіх унікальних слів, а тегів – відповідно індексами із словника тегів.

Дуже важливо, щоб при навчанні і при використанні навченої мережі використовувались одні і ті ж самі словники, тому хорошою практикою є

створення їх при навчанні і збереженні у окремі файли поруч з нейронною мережею. А потім при подальшій роботі відтворювати з існуючого файлу. Так, в результаті даної заміни, масив слів речення [«Студент», «Мельник», «навчається», «в», «КПІ»] буде у вигляді [0, 1, 2, 3, 4]. З списком розмічених сутностей для даного речення відбудеться аналогічна заміна: масив [«О», «ПЕРС», «О», «О», «ОРГ»] буде перетворений до вигляду [0, 1, 0, 0, 2]. Проте, це ще не всі перетворення, необхідні для навчання нейронної мережі.

Так як вже зазначалось вище, мережа буде давати ймовірність належності до кожного можливого тегу або існуючої сутності у навчальній вибірці. А на даному етапі просто індекс відповідної сутності з словника. Тому треба над мітками зробити операцію категоризації – розкласти кожну мітку в вектор нулів, і встановити одиницю лише для одного значення – якому відповідає мітка.

**Перевірка якості навченої нейронної мережі.** Важливою частиною в розробці продукту, в основі якого знаходиться модель машинного навчання чи нейронна модель – є перевірка якості. В даному випадку при оцінюванні рекурентної мережі (sequence model) також є декілька нюансів, про які треба згадати.

По-перше, в навчальній вибірці кількість прикладів з одним класом можна значно відрізнятись від кількості прикладів інших класів, внаслідок чого виникає дизбаланс. По-друге, іменована сутність може складатись з кількох токенів. Перша проблема вирішується завдяки правильному підбору метрики оцінювання. Згодом розглянемо які види метрик якості підходять для даного випадку.

Друга проблема вирішується правильним об’єднанням тегів на вищому рівні. Мається на увазі схема розмітки токенів BIO. Цей метод був описаний у першому розділі, його суть у додаванні префіксів «B-» (Begin) або «I-» (Inner). Таким чином всі спани іменованих сутностей будуть суцільні. І далі, отримавши прогноз від нейронної мережі, прибрати ці суфікси, такий чином

перейти до найвищого рівня можливих іменованих сутностей, наприклад, «ПЕРС» або «ОРГ».

Тепер повернемось до першої проблеми. Нехай у вхідному реченні всього 20 токенів, з яких 5 токенів – це іменовані сутності. Порахуємо, яка точність буде у випадку, коли мережа прогнозує, що в тексті взагалі немає сутностей. Тоді кількість правильно розмічених токенів буде 15, і відповідно точність такого прогнозу буде: $\frac{15}{20} * 100 = 75\%$. Досить непоганий результат, а у випадку, якщо іменованих сутностей взагалі не було б у тексті, або дуже маленька частина, то такий прогноз був би ще більш точним. Проте, насправді якість такої нейромережі дуже низька. Щоб подолати цю проблему, використовують для кожного класу наступні метрики: точність (precision), повноту (recall) та f1-score. А тепер спробуємо їх порахувати. Для цього визначимо необхідні показники:

- True positives – кількість міток класу, правильно передбачених
- False positives – кількість міток класу, неправильно передбачених
- False negatives – кількість міток, спрогнозованих як інший клас, проте насправді вони належать до того, для кого рахують даний показник

Тепер можна порахувати точність:

$$precision = \frac{TP}{TP + FP}$$

Повнота визначається як:

$$recall = \frac{TP}{TP + FN}$$

Тепер є всі значення для визначення f1-score як гармонічного середнього точності і повноти:

$$f1 - score = 2 * \frac{precision * recall}{precision + recall}$$

Якщо порахувати якість по метриці f1-score для прогнозу з всіх нулів, отримаємо результат – 0%, що відповідає справжній прогнозуючій здатності моделі.

## ВИСНОВКИ ДО РОЗДІЛУ 2

В даному розділі було розглянуто нейронні мережі. Детальніше розібрано будову основної одиниці нейромережі – нейрона, перші види нейронів – перцептрон та сигмоїдний нейрон. Також розглянуто алгоритм мінімізації функції похибки – градієнтний спуск, який використовується безпосередньо сам або його модифікації при навчанні більшості модель машинного навчання і нейронних мереж. Оскільки в даному проєкті буде відбуватись робота з текстовими послідовностями – прийнято рішення використовувати рекурентні нейронні мережі з типами нейронів LSTM. Тому також розглянуто особливості даного виду нейромереж і їхні особливості. Сформована архітектура мережі і зображена на відповідній схемі.

Також в даному розділі описано основний сценарій взаємодії користувача з готовим сервісом і всі етапи, через які проходять дані від моменту надходження в систему до повернення результату у відповідь.

Оскільки буде використовуватись нейронна мережа, спочатку її необхідно створити та навчити. У даному розділі було описано процес підготовки формування навчального корпусу для мережі та процес приведення прогнозів нейромережі у вигляд, зрозумілий для користувача, тобто, інтерпретація результату та перетворення ймовірностей до тегів з приставками BIO. А далі об’єднання отриманих тегів в неперервні теги конкретних іменованих сутностей.

В останньому пункті розділу було детально зображено процес оцінювання якості навченої моделі. Описано проблеми, які можуть виникати в робочі з послідовними даними та даними з дизбалансом класів. А також наведено приклади можливих розв’язків згаданих труднощів. Введено основну метрику оцінювання якості, яка буде використовуватись при аналізі нейронної мережі в даній роботі.

# РОЗДІЛ 3.
# РЕАЛІЗАЦІЯ СЕРВІСУ РОЗПІЗНАВАННЯ ІМЕНОВАНИХ СУТНОСТЕЙ

Розробка сервісу, описаного в даній роботі складається з кількох етапів: підготовка навчальних даних для нейронної мережі, тренування нейронної мережі, розгортання готового проєкту у вигляді веб-сервісу. Розберемо кожен з них детальніше.

## 3.1 Створення навчального корпусу

Як вже описувалось вище, навчальна вибірка формувалась з текстових даних, збалансованих по жанрах, з відповідними анотаціями, виконаними командою ентузіастів. Отже, було отримано 229 текстових файлів у двох розширеннях: .ann – анотації до відповідного тексту, .txt – текстовий документ. Детальніше можна побачити на Рис. 3.1. та Рис. 3.2.:

```
1  Зрозуміло , що український бізнес почав використовувати КСВ як інструме
2  З одного боку , саме через це більшість проектів КСВ здійснюються епізо
3  Винятком будуть хіба що представництва іноземних корпорацій .
```

Рис. 3.1. – Зображення текстового документу

```
1  T1      ОРГ 1137 1145   Життєлюб
2  T2      ПЕРС 1148 1168  Гаріка Корогодського
3  T3      РІЗН 1200 1250  Глобальному договору та Цілям сталого розвитку ООН
4  T4      ORG 1289 1301   Збройних Сил
```

Рис. 3.2. – Зображення файлу з анотаціями

На основі отриманих файлів було розроблено програмний код, який опрацьовує послідовно всі файли. В результаті формуються пари: «токен – мітка» для кожного слова у текстовому документі. На Рис. 3.3. зображено приклад розміченого фрагменту документа. Важливо відмітити, що залишається структура з виділеними реченнями в окремі масиви, оскільки для навчання нейронної мережі одним навчальним екземпляром буде одне речення.

```
[('Нещодавно', 'O'),
 ('на', 'O'),
 ('шаховій', 'O'),
 ('Олімпіаді', 'B-ORG'),
 ('українська', 'O'),
 ('жіноча', 'O'),
 ('збірна', 'O'),
 ('виборола', 'O'),
 ('бронзові', 'O'),
 ('нагороди', 'O'),
 (',', 'O'),
 ('а', 'O'),
 ('чоловіча', 'O'),
 ('-', 'O'),
 ('посіла', 'O'),
 ('шосте', 'O'),
 ('місце', 'O'),
 ('із', 'O'),
 ('понад', 'O'),
 ('півтори', 'O'),
 ('сотні', 'O'),
 ('команд', 'O'),
 ('країн', 'O'),
 ('планети', 'O'),
 ('.', 'O')]
```

Рис. 3.3. – Приклад розміченого речення

### 3.2 Обробка навчальних даних

Отримавши розмічену вибірку, ще не можна було навчати нейронну мережу. Адже спочатку тренувальний корпус необхідно опрацювати належним чином.

Першим кроком було створення підключення до бази даних з ВЕСУМ. Оскільки в якості бази даних було використано MongoDB, для реалізації був реалізований клас, за допомогою якого можна було робити запити до електронного словника – «Vesum_interface». Головним призначенням ВЕСУМа в даному випадку було приведення слова до його нормальної форми, що й було реалізовано в функції «get_main_form_from_vesum». Дана функція приймає на вхід слово або токен з тексту, шукає всі записи в електронному словнику, визначає нормальні форми, якщо для всіх знайдених записів вони однакові – повертає її, в разі наявних різних варіантів – повертає вхідне слово.

Дана функція створена для зменшення кількості унікальних слів, а саме завдяки виникненню різних форм при відмінюванні слів, можливості бути в

однині або множині, в різних особах і т. п. Наприклад, у тексті може бути слово «року» - слово «рік» у родовому відмінку або слово «рок» у родовому відмінку. Нормальна форма – різна, слова різні по значенню, тому дану заміну робити не можна. Тому, реалізувавши даний коннектор до бази даних з необхідною логікою, далі було створено список всіх слів корпуса. Із навчальної вибірки, яка вже була токенізована по реченнях і по словах, відбувається перебір кожного елементу, із застосуванням функції знайдення нормальної форми слова, а результат виконання даної функції додався до списку всіх слів – «words». Далі з нього видаляються дублікати, додаються службові слова «ENDPAD» та «UNKNOWN», і, таким чином, було отримано список всіх слів навчального корпуса.

Наступним кроком була підготовка словника з векторними представленнями слів. Для цього використався навчений word2vec, попередньо завантажений. В результаті створено словник «word_embedding» з індексами слів і векторними представленнями.

Нейронна мережа працює з вхідними даними однакових розмірів. З Рис. 3.4. було визначено оптимальне значення:

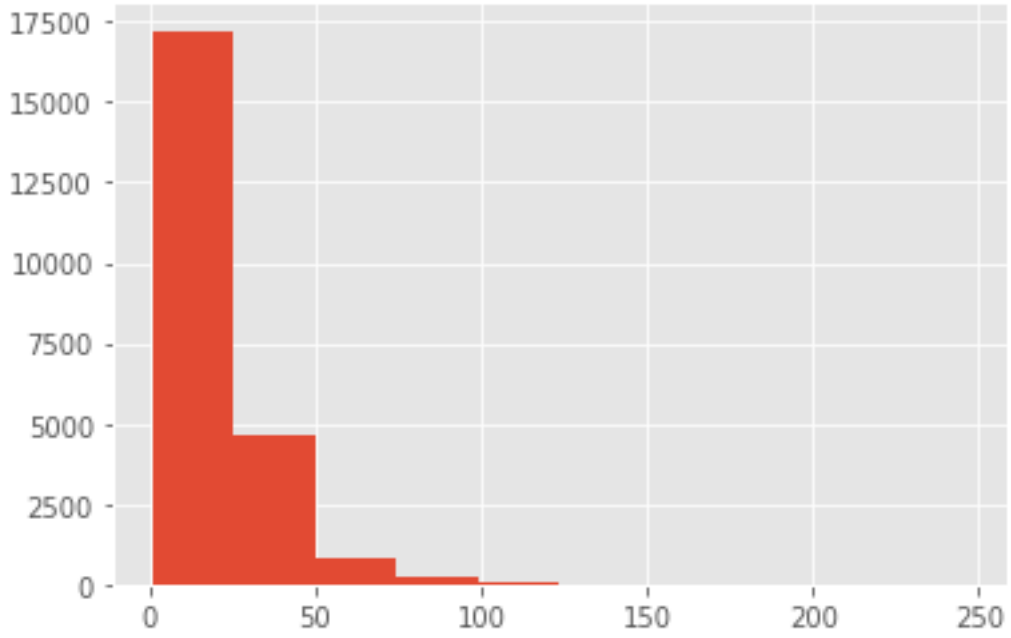



Рис. 3.4. – Розподіл довжин речень у корпусі

Встановивши довжину речення 70 токенів, прийдеться лише 381 речення зі всього документу (22942 речення) зменшувати, тобто, залишати лише перші 70 елементів, всі інші – відкинути. Речення, в яких кількість токенів менша – наповнюються спеціальним словом «ENDPAD» з додаванням до кінця. В результаті даної операції речення буде виглядати, як на Рис. 3.5.

```
array(['Прикметно', ',', 'що', 'період', 'польський', 'панування', 'у',
       'Львові', ',', 'який', 'початися', 'від', 'середина', 'XIV',
       'століття', ',', 'знаменуватися', 'засилля', 'не', 'поляк', ',',
       'а', 'німець', 'у', 'всіх', 'сфера', 'міський', 'життя', '.',
       'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD',
       'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD',
       'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD',
       'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD',
       'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD',
       'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD', 'ENDPAD'], dtype='<U13')
```

Рис 3.5. – Речення, вирівняне до довжини в 70 слів

Так як теги ставляться для кожного слова, тому їх для вирівняних речень до однакової довжини теж необхідно додавати, якщо речення було меншої довжини, проте в даному випадку не доставляти токен «ENDPAD», а тег «O», який означає, що відповідне слово не позначає ніяку сутність. Приклад переробленої анотації для речення з корпуса зображено на Рис. 3.6.

```
array(['O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-LOC', 'O', 'O', 'O', 'O',
       'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
       'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
       'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
       'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
       'O', 'O', 'O', 'O', 'O', 'O'], dtype='<U5')
```

Рис. 3.6. – Анотація іменованих сутностей для вирівняного речення

Наступним кроком є створення словників – «word2indx» та «tag2indx». З назви можна зрозуміти, що вони виконують функцію заміни слова на його індекс, і мітки анотації на її індекс, відповідно. Це відбувається наступним чином: в словнику ключем являється слово, а значенням – індекс (унікальне число, яке ідентифікує дане слово). Така ж логіка і для формування словника для іменованих сутностей. Сформувавши дані структури, можна легко закодувати текстові дані у числові значення, з якими математична модель зможе працювати. Тому, проводиться заміна кожного токена текстового

документа та тегів іменованих сутностей. Приклад закодованого речення на Рис. 3.7., а закодованої анотації для цього речення – на Рис. 3.8.

```
array([11129, 26288,  2352, 30231, 16585, 26156, 24353, 11706, 26288,
       21131, 11292,  4965, 18777, 23888, 29182, 26288,  2026, 11370,
       24677, 27959, 26288, 21764, 14179, 24353, 16543, 30463, 15058,
       17427, 27885, 30901, 30901, 30901, 30901, 30901, 30901, 30901,
       30901, 30901, 30901, 30901, 30901, 30901, 30901, 30901, 30901,
       30901, 30901, 30901, 30901, 30901, 30901, 30901, 30901, 30901,
       30901, 30901, 30901, 30901, 30901, 30901, 30901, 30901, 30901,
       30901, 30901, 30901, 30901, 30901, 30901, 30901], dtype=int32)
```

Рис. 3.7. – Закодоване речення

```
array([6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 3, 8, 6, 6, 6, 4, 5, 6,
       6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
       6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
       6, 6, 6, 6], dtype=int32)
```

Рис. 3.8. – Закодована анотація

Проте, для анотацій необхідно провести ще одне перетворення – категоризацію. Ідея заключається в тому, що на даному етапі мітки іменованих сутностей закодовані у відповідні ідентифікатори, проте у вигляді порядкової (ordinal) змінної. Але це незалежні категорії, тому необхідно передати для нейронної мережі у вигляді, де кожен індекс заміняється на вектор, в якому всі елементи нулі, окрім відповідного індекса тега, для якого ставиться одиниця. Приклад категоризованих анотацій на Рис. 3.9.

```
array([[0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
```

Рис. 3.9. – Категоризовані анотації

І останній крок у підготовці даних – це розбиття корпусу на тренувальну та тестову вибірки. Це робиться для того, щоб залишити певну частину даних, справжні результати якої ми знаємо, невідомою для мережі. Так можна буде дізнатись показники якості не лише для даних, на яких навчалась модель, але і на нових, не бачених при навчанні. Іншими словами, так можна перевірити чи не відбулось перенавчання, виявити чутливість моделі і її узагальнюючу здатність. Для цього, використовуючи функцію train_test_split, розділяємо текстовий корпус і відповідні анотації на дві частини у співвідношенні 90 % для навчання та 10 % для тестування зі збереженням співвідношення кількості тегів. Це виконується для підтримання балансу класів у навчальній і тестовій вибірках, що дозволить максимально зберегти схожість обох. В результаті було отримано X_train, X_test – навчальна і тестова вибірки, y_train, y_test – відповідні анотації, приведені до необхідного вигляду. Розмірності отриманих об’єктів наведені у Таблиці 3.1.

*Таблиця 3.1.*

**Розмірності навчальних і тестових даних**

| Об’єкт | Розмірність |
|---|---|
| X_train | (20647, 70) |
| X_test | (20647, 70, 9) |
| y_train | (2295, 70) |
| y_test | (2295, 70, 9) |

### 3.3 Навчання нейронної мережі

Реалізація нейронної мережі виконана за допомогою бібліотеки Keras. Keras – бібліотека для роботи з нейронними мережами, написана на Python. Це обгортка над фреймворками Deeplearning4j, TensorFlow, Theano. Основні переваги даного інструмента наступні.

Перш за все – це зручність у користуванні. Це API, розроблене для людей, а не машин. Тому, користувацький досвід приділяється увагою найбільше. Зменшено кількість дій, необхідних для випадків загального використання, забезпечується чіткий і дієвий зворотній зв'язок при виникненні помилок.

По-друге, модульність. Нейромережа розглядається як набір автономних, повністю конфігурованих модулів, які можна підключати разом з якомога найменшими обмеженнями. Наприклад, шари нейронної мережі, функції похибок, схеми ініціалізації, функції активації та засоби регуляризації – це окремі модулі, які можна комбінувати для створення нових моделей. По-третє, масштабованість. Нові модулі легко додавати (у вигляді класів або функцій), існуючі модулі надають достатньо прикладів. Можливість легко розширяти набір необхідних засобів надає повну виразність, що робить Keras придатним для поглиблених досліджень.

Ще однією перевагою Keras є робота з Python. Не потрібно використовувати додаткових файлів конфігурацій моделей у декларативному форматі. Нейронні мережі описуються на мові Python, що є дуже компактно, легко у виявленні та виправленні помилок і дозволяє швидко реалізовувати індивідуальні модулі.

Архітектура нейронної мережі, реалізована у даній роботі наступна. Вхідних шар – очікує на вхід 70 елементів (оскільки в робочі прийнято рішення всі речення вирівняти до довжини 70 слів) – індексів слів, використаних з словника «word2indx». Далі шар з векторним представленням слів – embedding. В ході навчання нейромережі він буде донавчати свої параметри представлень слів, також можна вибирати чи навчати з нуля або використати готовий попередньо навчений - «word_embedding». Але мінус готового, що вихідна розмірність кожного слова – 300, що може бути забагато, у випадку невеликої кількості навчальних даних. Тому, при навчанні було випробувано різні комбінації, що буде представлено у таблиці результатів нижче.

Наступний шар – Dropout. Основна функція даного шару – регуляризація. При навчанні кожен нейрон виключається з мережі з ймовірністю, яка задається при ініціалізації, в даному випадку – 10 %. Виключення нейрона означає, що при будь-яких вхідних даних він завжди буде повертати 0. Наступний шар мережі – двонаправлений LSTM шар, головним параметром якого є кількість нейронів. Він виконує основну роль у нейромережі, вивчаючи всі контекстно-залежні властивості елементів у вхідних послідовностях. Саме завдяки даному шарові це можна вважати рекурентною мережею.

Далі знаходиться останній, вихідний шар моделі. Це повнозв'язний шар, який складається з кількості нейронів, яка дорівнює кількості можливих класів - 9. Застосовуючи softmax функцію активації, на виході отримуються значення в межі [0, 1] які можна сприймати як ймовірність належності до відповідного класу, і тому відносити даний приклад до того класу, в якого найбільше дане значення. Остання дія перед навчанням моделі – скомпілювати її, задаючи параметри: оптимізатор – "adam", функція помилок – «categorical_crossentropy», метрика оцінювання якості моделі – точність («accuracy»). Далі відбувались експерименти з навчання нейронної мережі з різними параметрами, детальніше можна побачити у Таблиці 3.2.

*Таблиця 3.2.*

**Експерименти з навчання нейронної мережі**

| Параметри мережі | f-score на навчальній вибірці | f-score на тестовій вибірці |
|---|---|---|
| Embedding_dim=50<br>Use_pretrained_embedding=False<br>LSTM_units=100<br>Batch_size=32; Use_main_form=False | 95.3 % | 80.4 % |

*Продовження таблиці 3.2.*

| Параметри мережі | f-score на навчальній вибірці | f-score на тестовій вибірці |
|---|---|---|
| Embedding_dim=50<br>Use_pretrained_embedding=False<br>LSTM_units=50<br>Batch_size=32; Use_main_form=False | 91.4 % | 75.5 % |
| Embedding_dim=50<br>Use_pretrained_embedding=False<br>LSTM_units=50<br>Batch_size=64; Use_main_form=False | 94.58 % | 81.8 % |
| Embedding_dim=50<br>Use_pretrained_embedding=False<br>LSTM_units=32<br>Batch_size=64; Use_main_form=True | 94.95 % | 83.7 % |
| Embedding_dim=300<br>Use_pretrained_embedding=True<br>LSTM_units=32<br>Batch_size=64; Use_main_form=True | 61 % | 50 % |
| Embedding_dim=20<br>Use_pretrained_embedding=False<br>LSTM_units=32<br>Batch_size=32; Use_main_form=True | 55 % | 50.9 % |
| Embedding_dim=50<br>Use_pretrained_embedding=False<br>LSTM_units=32<br>Batch_size=64; Use_main_form=True | 95.49 % | 88.95 % |

Як видно з таблиці експериментів, найвищу якість на тестовій вибірці здобула нейронна мережа з конфігурацією з останнього рядка. Значна різниця у якості прогнозування на навчальній і тестовій вибірках говорить про перенавчання моделі. Дана проблема вирішується шляхом регуляризації – яку можна здійснити двома шляхами: додавання шару Dropout або збільшенням навчальної вибірки. Другий спосіб перевірити не було спроби, адже було залучено 90% всіх даних, більше розмічених текстових корпусів не було доступно. Спроби додавати шари регуляризації зменшували якість моделі як на навчальній, так і на тестовій вибірках, тому в якості основної мережі був використаний варіант з найвищою якістю прогнозу на тестовій вибірці. На Рис. 3.10. зображено якість нейромережі по метриці accuracy, яка використовувалась при навчанні на навчальній і валідаційній (відокремлена автоматично під час навчання для перевірки якості мережі на невідомих даних) вибірці.

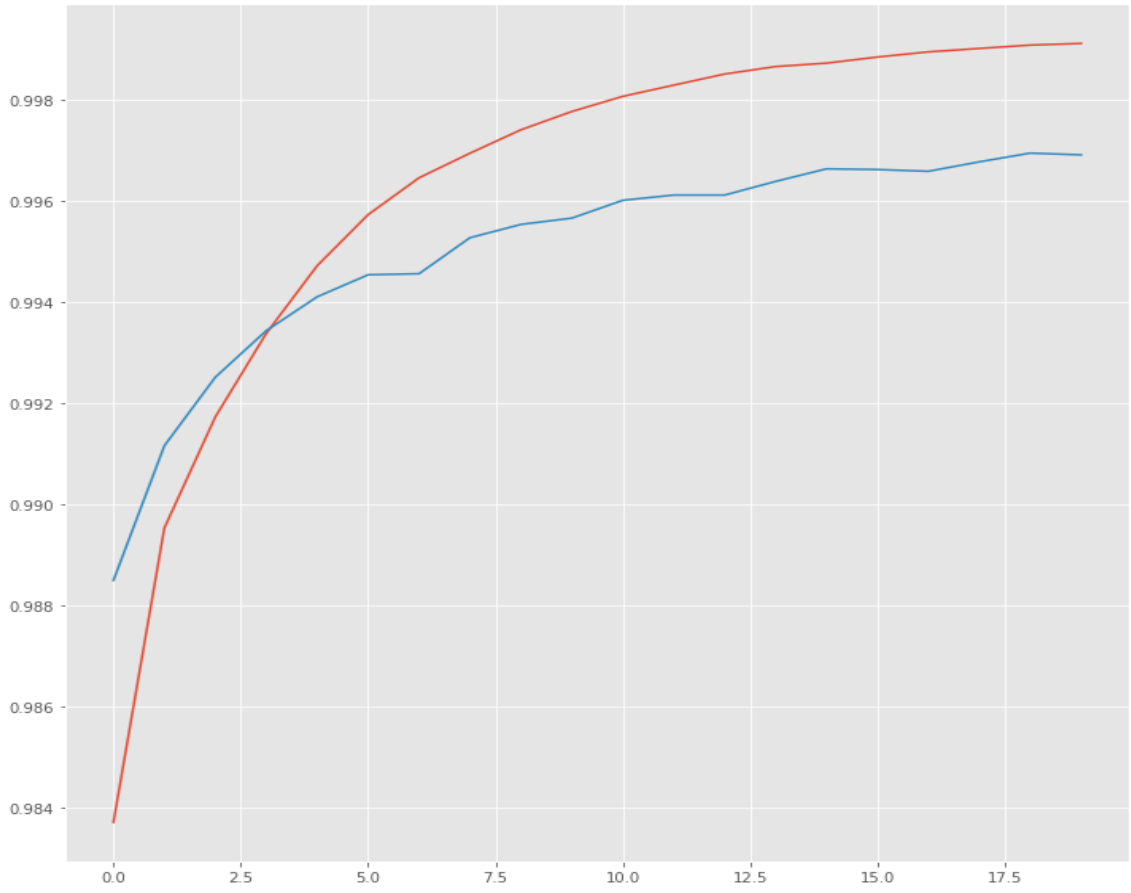



Рис. 3.10. – Точність нейронної мережі на тренувальній (червона крива) і валідаційній (синя крива) вибірках на різних епохах навчання

Також, використовуючи функцію classification_report, визначимо значення по метриках precision, recall та f-score по кожному класові (типові іменованих сутностей) у прогнозі на тестовій вибірці.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| PERS | 0.93 | 0.91 | 0.92 | 778 |
| LOC | 0.90 | 0.84 | 0.87 | 288 |
| ORG | 0.86 | 0.78 | 0.82 | 147 |
| MISC | 0.90 | 0.69 | 0.78 | 108 |
| micro avg | 0.91 | 0.86 | 0.89 | 1321 |
| macro avg | 0.91 | 0.86 | 0.89 | 1321 |

Рис. 3.11. – Оцінювання якості прогнозу нейронної мережі по кожному типові іменованої сутності

Останнім кроком зберігаємо натреновану нейронну мережу у відповідний файл. Це можна зробити, використовуючи функцію «save» бібліотеки Keras, отримавши об’єкт з розширенням .h5. Також, зберігаємо об’єкти «word2indx» і «tag2indx», які необхідні для обробки вхідного текстового документу та інтерпретації прогнозу нейронної мережі.

### 3.4 Застосування мережі

Отримавши збережену нейронну мережу і всі необхідні об’єкти для обробки вхідного речення, наступним кроком необхідно створили засіб для ефективного застосування мережі на нових текстових документах. Спочатку було створено функцію, яка приймає на вхід текст і повертає масив знайдених іменованих сутностей. Розберемо, що відбувається у ній детальніше.

Першим кроком перевіряється наявність всіх необхідних об’єктів: інтерфейс для роботи з ВЕСУМ, навчена нейронна мережа, словник для перетворення токенів у відповідні індекси і навпаки та словник для перетворення індексів прогнозованих тегів сутностей у відповідні теги. У випадку відсутності – відбувається ініціалізація об’єкта (зчитування з директорії saved_objects). Далі вхідний текстовий корпус обробляється аналогічно як при процесі навчання. Відбувається токенізація, вирівнювання кожного речення до кількості 70 токенів та приведення до нормально форми

всіх можливих слів. Далі – заміна слів на їхні індекси зі словника всіх слів, а у випадку відсутності слова – воно заміниться на індекс слова «UNDEFINED».

Закодовані послідовності передаються до нейронної мережі для прогнозу. На виході отримуються прогнози для кожного токена кожного речення у вигляді ймовірності належності до кожного можливого класу. Для переходу від ймовірностей до класів – використовується функція знаходження індекса максимального значення у послідовності. На даному етапі отримано список належності до класу кожного токену, проте ще необхідно розкодувати дані індекси до зрозумілих позначень іменованих сутностей. Саме для цього і використовується створений при навчанні словник класів з відповідними індексами. Наступний крок – виділення іменованих сутностей з отриманих прогнозів. Отримавши прогнозовані класі для кожного токену, можна з них виділити всі отримані іменовані сутності, які будуть мати наступні характеристики: тип сутності, стартова позиція у вхідному тексті, позиція закінчення, фрагмент з тексту з відповідною сутністю. Для цього створюються словники з переліченими атрибутами, які по своїй структурі нагадуються json. Отже, в результаті на кожну знайдену іменовану сутність створюється словник, який її описує. Функція прогнозування повертає список таких об’єктів у відповідь.

Другий етап підготовки сервісу – це реалізація можливості застосунку даної нейронної мережі у вигляді веб-сервісу. Це відбувається за наступним сценарієм. Відправляється POST запит, в тілі якого передається необхідний текст. Далі даний текст передається до вищеописаної функції прогнозування. А отриманий список знайдених іменованих сутностей повертається у відповідь на запит користувачу. Дана можливість реалізована за допомогою бібліотеки Flask, яка і обробляє сценарій отримання описаного запиту.

### 3.5 Тестування отриманих результатів

Хоча ще на етапі навчання нейронної мережі було відкладено тестову вибірку, на якій можна було оцінити якість прогнозування по кожному типі іменованої сутності, необхідно приділити увагу для очного розгляду прикладів

роботи створеного алгоритму. Це забезпечить краще розуміння властивостей мережі, знаходження сильних і слабких сторін, можливих причин помилок та способи їх усунення і подальшого покращення як самої якості прогнозів нейронної мережі, так і роботу всього сервісу в цілому. Оскільки в даній роботі використовуються іменовані сутності: Персона, Локація, Організація та Інші, розглянемо приклади роботи з кожним з них. Також, наведемо результати прогнозування, яке базується на rule-based підході.

*Таблиця 3.3.*

**Прогнозування персон у тексті**

| Вхідний текст | Результат нейронної мережі | Результат rule-based підходу |
|---|---|---|
| І тоді Макар придумав хитрість. | [{ "entity_type": "PERS", "start_index": 7, "finish_index": 12, "text_entity": "Макар" }] | [{"type": "Person", "fact": {"name": {"first": "макар"}}, "span": [7, 12] } ] |
| І тільки вперто заплющував очі, коли до палати входили Нані чи Новаковський. | [{ "entity_type": "PERS", "start_index": 55, "finish_index": 59, "text_entity": "Нані"}, { "entity_type": "PERS", "start_index": 63, "finish_index": 75, "text_entity": "Новаковський" }] | [ {"type": "Person", "fact": {"name": { "first": "коли"}}, "span": [ 32, 36]}, { "type": "Person", "fact": {"name": { "last": "новаковський"}}, "span": [63, 75]}] |
| І поїхав до відомого космацького майстра Ф. Кравчука. | [{ "entity_type": "PERS", "start_index": 41, "finish_index": 52, "text_entity": "Ф. Кравчука" }] | [{ "type": "Person", "fact": {"position": "майстра","name": {"first": "Ф", "last": "кравчук"}}, "span": [33, 52]}] |

*Таблиця 3.4.*

**Прогнозування локацій у тексті**

| Вхідний текст | Результат нейронної мережі | Результат rule-based підходу |
|---|---|---|
| На мості Патона притулився до огорожі, очі у воду. | [{ "entity_type": "LOC", "start_index": 3, "finish_index": 15, "text_entity": "мості Пато на" }] | [] |
| Прага подобалася Роману. | [{ "entity_type": "LOC", "start_index": 0, "finish_index": 5, "text_entity": "Прага"}, {"entity_type": "PERS", "start_index": 17, "finish_index": 23, "text_entity": "Роману"}] | [{ "type": "Location", "fact": {"name": "прага"}, "span": [0, 5] }] |
| Уранці вони нагадали Романові вхід до Личаківського цвинтаря. | [{ "entity_type": "PERS", "start_index": 21, "finish_index": 29, "text_entity": "Романові"} , {"entity_type": "LOC", "start_index": 38, "finish_index": 60, "text_entity": "Личаківськ ого цвинтаря"}] | [{"type": "Location", "fact": { "name": "роман" }, "span": [21, 29]} ] |

*Таблиця 3.5.*

**Прогнозування організацій у тексті**

| Вхідний текст | Результат нейронної мережі | Результат rule-based підходу |
|---|---|---|
| Приклеєний до спини погляд допровадив Аліну до дверей Академії. | [{ "entity_type": "ORG", "start_index": 55, "finish_index": 63, "text_entity": "Академії"}] | - |
| Обласна рада звертається з поданням до Верховної Ради України щодо зміни меж. | [{ "entity_type": "ORG", "start_index": 8, "finish_index": 12, "text_entity": "рада" }] | - |
| Коли 2004-го з «Океану Ельзи» пішли троє учасників. | [{ "entity_type": "ORG", "start_index": 16, "finish_index": 28, "text_entity": "Океану Ельзи" }] | - |

*Таблиця 3.6.*

**Прогнозування інших сутностей у тексті**

| Вхідний текст | Результат нейронної мережі | Результат rule-based підходу |
|---|---|---|
| Недарма ж за гороскопом була Рибами. | [{ "entity_type": "MISC", "start_index": 29, "finish_index": 35, "text_entity": "Рибами"}] | - |

*Продовження таблиці 3.6.*

| Вхідний текст | Результат нейронної мережі | Результат rule-based підходу |
|---|---|---|
| Відносини, що регулюються цим Законом. | [{ "entity_type": "MISC", "start_index": 30, "finish_index": 37, "text_entity": "Законом"}] | - |
| Під’їжджає скромний чорненький «Ягуар». | [ { "entity_type": "MISC", "start_index": 33, "finish_index": 38, "text_entity": "Ягуар"}] | - |

З таблиць порівнянь можна зробити висновок, що нейронна мережа має досить високі результати у прогнозуванні. Перший тип сутностей – персона. Серед наведених прикладів нейромережа спрогнозувала все правильно, в той час як підхід, заснований на правилах допустив одну помилку – хибно ідентифікував слово «коли» як персону у другому прикладі. Як і видно з результатів нейронної моделі на тестовій вибірці – показник по метриці f-score – 92%, що і підтверджується на отриманих прогнозах.

Наступна таблиця з прикладами результатів – по локаціях. Знову ж таки, нейромережа влучно відгадала всі локації і виявила правильно персону. Ручний підхід помилково дану сутність відмітив як локацію (слово «Роман» з другого прикладу), а також не виявив всі наявні сутності, наприклад, «міст Патона». Серед прикладів роботи з типами – організація, нейромережа допустила помилки у другому прикладі, відмітивши «рада» як локація, правильна відповідь у даному випадку – «Обласна рада», а також пропустила «Верховної Ради України», що теж являється іменованою сутністю – організацією. Ручний підхід не передбачає виявлення даного типу сутностей.

Тепер розглянемо результати роботи по типу – Інше. Нагадаю, що це іменовані сутності, але які не входять до попередніх класів. Нейронна мережа правильно спрогнозувала всі приклади, але, спробувавши вказати інші знаки зодіаку в першому реченні, вона не виявляла сутності, що може свідчити про відсутність узагальнюючої властивості для даного випадку. Також було проведено експерименти з третій прикладом, заміняючи модель машини «Ягуар» на інші, з чим мережа успішно впоралась і правильно спрогнозувала дане слово як Інший тип сутності. Ручний підхід для даних сутностей теж не має реалізації.

Проаналізувавши отримані результати, можна сказати, що нейронна мережа має досить хороші результати у прогнозуванні іменованих сутностей. Однією з переваг є наявність класифікації 4 можливих типів сутностей. Також, завдяки можливості враховувати контекст з тексту, вона може досить влучно працювати з новими словами та випадками, які були мало або частково розглянуті при навчанні. Проте, існують також слабкі сторони. Нейронна мережа, як і інші моделі машинного навчання, дуже вразливі до навчальної вибірки. Так як в для даної мережі був дизбаланс класів у вибірці, вона змогла краще навчатись розрізняти ті типи, які частіше зустрічались при навчанні, і відповідно, гірше класифікує сутності у випадках, які були в недостатній кількості при навчанні. Тому, хорошою практикою є періодичне донавчання мережі, що забезпечить актуальність і стійкість мережі.

Реалізований сервіс розміщений у GitHub за посиланням: https://github.com/ValentynKharchuk/NER_UA. Там можна знайти директорію зі збереженими об’єктами (saved_objects): натренованою нейромережею, словником унікальних слів і тегів, а також навчальний корпус (директорія data_prepared) та jupyter ноутбуки із обробкою даних, тренуванням мережі, створенням бази даних із ВЕСУМ та реалізацією можливості розпізнавання іменованих сутностей у вигляді веб-сервіса.

## ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі було детально розглянуто процес реалізації сервісу з розпізнавання іменованих сутностей в українській мові. Для вирішення даної задачі використовувалась нейронна мережа. Тому, можна виділити головні етапи при реалізації.

Перш за все – це формування навчальної вибірки. З цією ціллю використано корпус розмічених україномовних текстів у різноманітних жанрах. Наступним етапом стала підготовка вибірки для навчання нейронної мережі. Над отриманими текстами відбувались операції по обробці, перетворенні даних у необхідний вигляд. Далі описано ініціалізацію нейронної мережі та процес навчання. Зображено графіки якості прогнозувань на різних етапах навчання, також наведена таблиця по метриках f-score, precision, recall по кожному типові іменованих сутностей. Наступним етапом була розробка можливості прогнозування нейронної мережі у нових текстах, зі всіма необхідними перетвореннями тексту і формуванням результату у вигляді списку json об’єктів для кожної знайденої сутності. Дана взаємодія із нейромережею реалізована у виглядів веб-сервісу, що також описано у даному розділі.

Отримавши готовий сервіс для пошуку іменованих сутностей, було проведено експерименти з прогнозування. Результати експериментів наведені у відповідних таблицях, також для порівняння є результати виконання rule-based підходу. Проведено аналіз помилок. Наведені переваги та недоліки даного способу виявлення іменованих сутностей у тексті, а також запропоновані варіанти для подальшої підтримки актуальності нейронної мережі та покращення результатів прогнозування.

# ВИСНОВКИ

В результаті даного проєкту було розроблено сервіс з розпізнавання іменованих сутностей для україномовних текстів. Сервіс можна використовувати для корекції помилок попередньо розробленої системи, в результаті якої знаходиться rule-based підхід. Перевагою даного підходу є використання нейронних мереж для прогнозування належності до одного з можливих типів іменованих сутностей кожного слова тексту. Також даний сервіс можна використовувати у вигляді веб-сервісу, надсилаючи необхідні запити і отримуючи у відповідь список знайдених сутностей. Інший варіант застосування – інтеграція у вигляді модуля розподіленої системи аналітичного опрацювання даних.

У першому розділі проведено огляд існуючих рішень у даній області, розглянуто основні класи задач NLP, основні засоби для вирішення задач з обробки природньої мови та їх складності. Також детально розглянута задача розпізнавання іменованих сутностей. Наведено приклади можливих вирішень, складнощі, які виникають при вирішенні задачі та доведено актуальність даної класу задач для текстів на українській мові.

У другому розділі наведено математичне обгрунтування використання саме нейронних мереж для прогнозування. Детально розглянуто перцептрон, рекурентні нейронні мережі та процес їхнього навчання. Також спроєктовано сервіс, який було розроблено в результаті даної роботи. Описано послідовність операцій, які будуть виконуватись над даними з моменту надходження до системи і до повернення відповіді користувачеві.

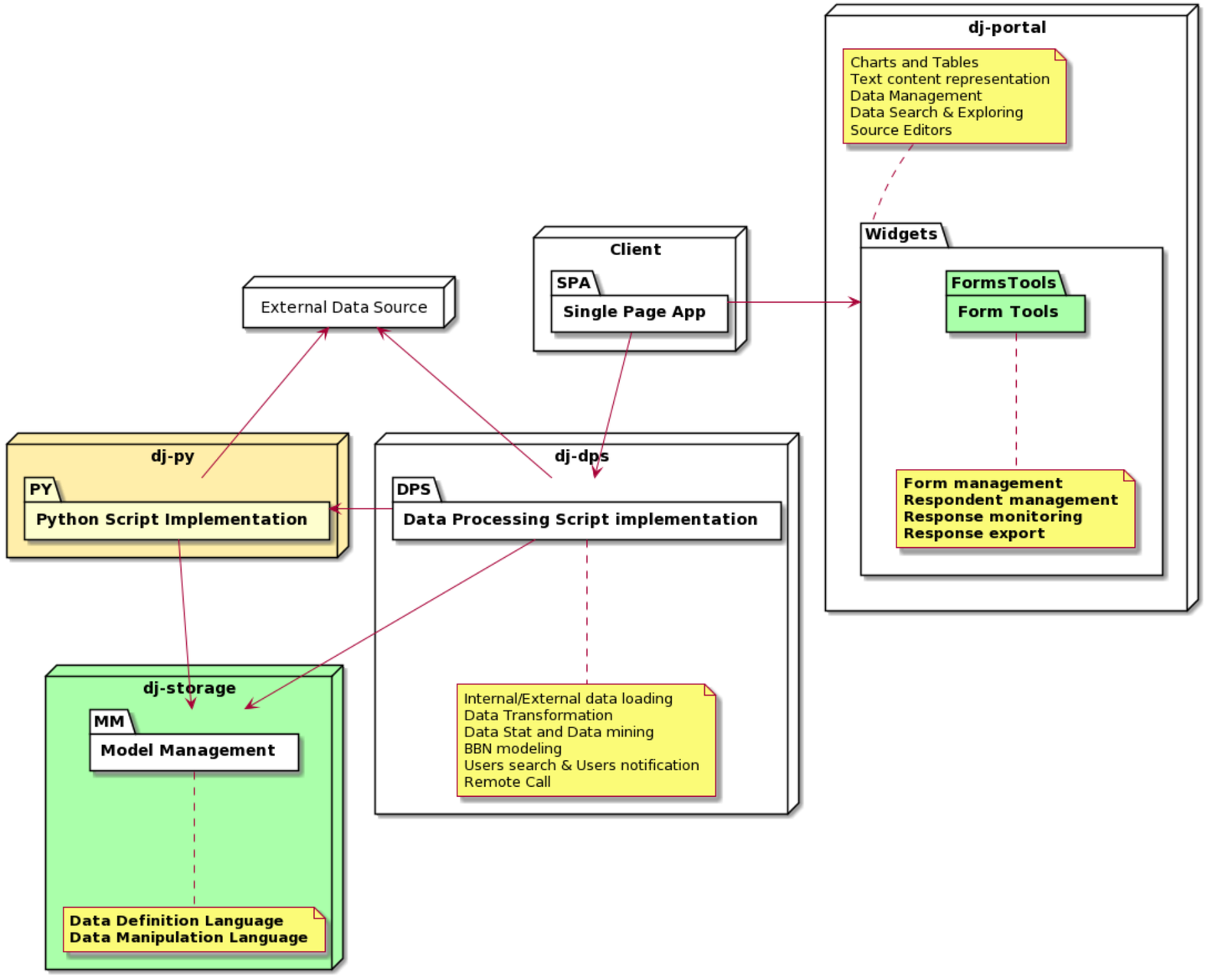
У третьому розділі продемонстровано реалізацію описаного сервісу, а саме такі етапи як: підготовка навчальних даних, тренування нейронної мережі, створення функції для зручного прогнозування на вхідних даних. Також продемонстровано роботу розробленого сервісу, порівняння результатів із rule-based підходом, проаналізовано переваги і недоліки даного підходу.

## ПЕРЕЛІК ПОСИЛАНЬ

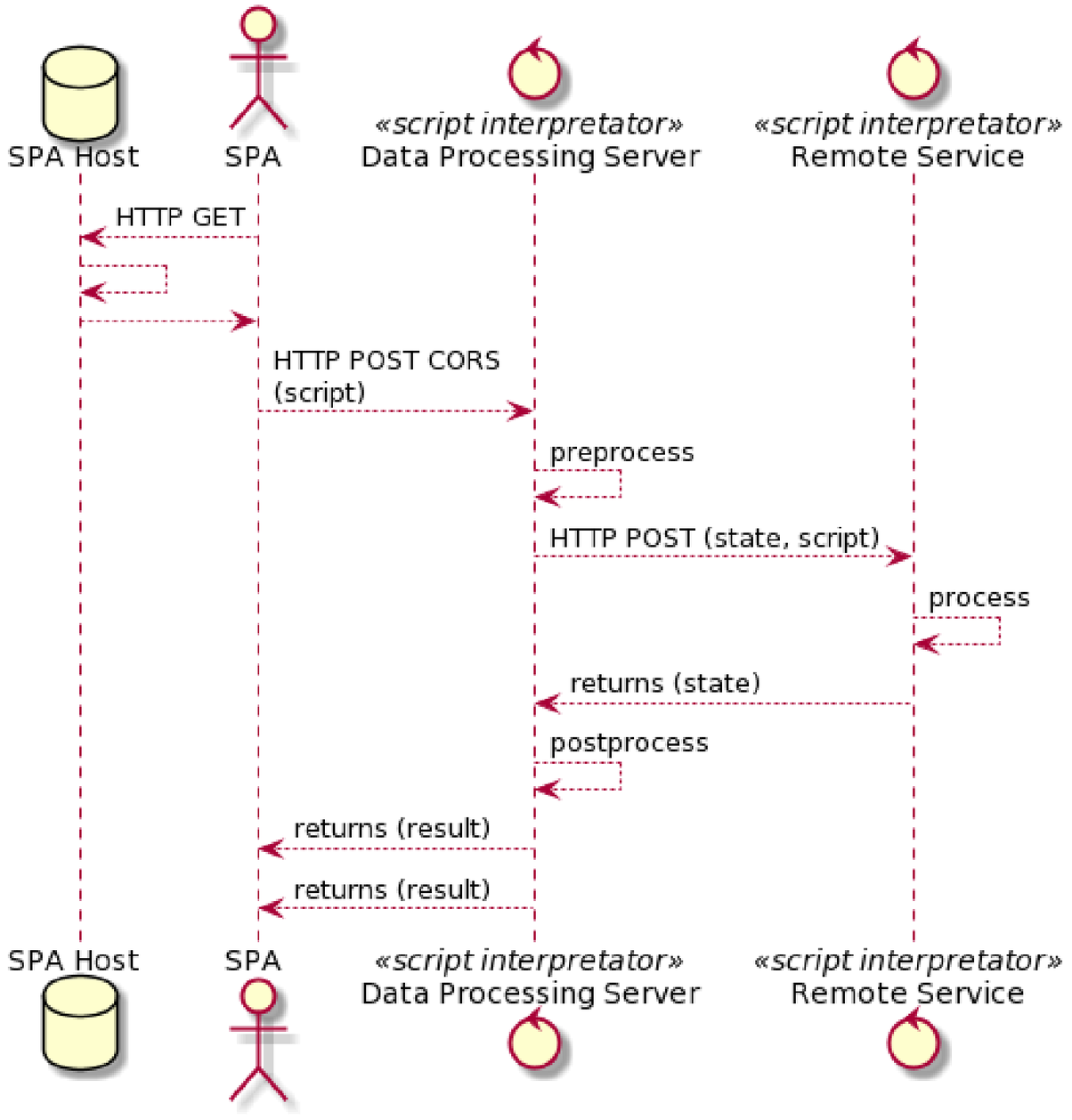

1. «Your guide to Natural Language Processing(NLP)» [Електронний ресурс]: [Веб-сайт]. – Режим доступу до ресурсу: https://towardsdatascience.com/your-guide-to-natural-language-processing-nlp-48ea2511f6e1 (дата звернення 10.05.2020).
2. «Amazon Comprehend Medical» [Електронний ресурс]: [Веб-сайт]. – Режим доступу до ресурсу: https://aws.amazon.com/ru/comprehend/medical/ (дата звернення 10.05.2020).
3. «Watson Personality Insights» [Електронний ресурс]: [Веб-сайт]. – Режим доступу до ресурсу: https://www.ibm.com/cloud/watson-personality-insights (дата звернення 10.05.2020).
4. «The Limitations of Stylometry for Detecting Machine-Generated Fake News» [Електронний ресурс]: [Веб-сайт]. – Режим доступу до ресурсу: https://arxiv.org/pdf/1908.09805.pdf (дата звернення 10.05.2020).
5. *«Distributional Structure»* [Електронний ресурс]: [Веб-сайт]. – Режим доступу до ресурсу: *10.1080/00437956.1954.11659520* (дата звернення 10.05.2020).
6. Jones K. S. «A statistical interpretation of term specificity and its application in retrieval» Journal of Documentation: журнал. — MCB University: MCB University Press, 2004. — Т. 60, № 5. — С.493-502.
7. «Tokenization» [Електронний ресурс]: [Веб-сайт]. – Режим доступу до ресурсу: https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html (дата звернення 10.05.2020).
8. «Stemming and lematization» [Електронний ресурс]: [Веб-сайт]. – Режим доступу до ресурсу: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html (дата звернення 10.05.2020).

9. Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg; Dean, Jeffrey «Distributed Representations of Words and Phrases and their Compositionality», 2013.
10. «Latent Dirichlet Allocation» [Електронний ресурс]: [Веб-сайт]. – Режим доступу до ресурсу: https://web.archive.org/web/20120207011313/http://jmlr.csail.mit.edu/papers/volume3/blei03a/blei03a.pdf (дата звернення 10.05.2020).
11. «Introduction to Named Entity Recognition» [Електронний ресурс]: [Веб-сайт]. – Режим доступу до ресурсу: https://cs.nyu.edu/grishman/NEtask20.book_2.html#HEADING1 (дата звернення 10.05.2020).
12. «Великий електронний словник української мови (ВЕСУМ)» [Електронний ресурс]: [Веб-сайт]. – Режим доступу до ресурсу: https://github.com/brown-uk/dict_uk/blob/master/doc/announcement.md (дата звернення 10.05.2020).
13. «lang-uk» [Електронний ресурс]: [Веб-сайт]. – Режим доступу до ресурсу: https://lang.org.ua/en/corpora/ (дата звернення 10.05.2020).
14. Rosenblatt, Frank «The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain», 1958, Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386-408.
15. Yuan, Ya-xiang, «Step-sizes for the gradient method», *1999, p.785.*
16. Miljanovic, Milos, «Comparative analysis of Recurrent and Finite Impulse Response Neural Networks in Time Series Prediction», 2012, Indian Journal of Computer and Engineering.
17. Sepp Hochreiter*;* Jürgen Schmidhuber, «Long short-term memory», 1997, pp.1735–1780.
18. Dey, Rahul; Salem, Fathi M., «Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks», 2017.

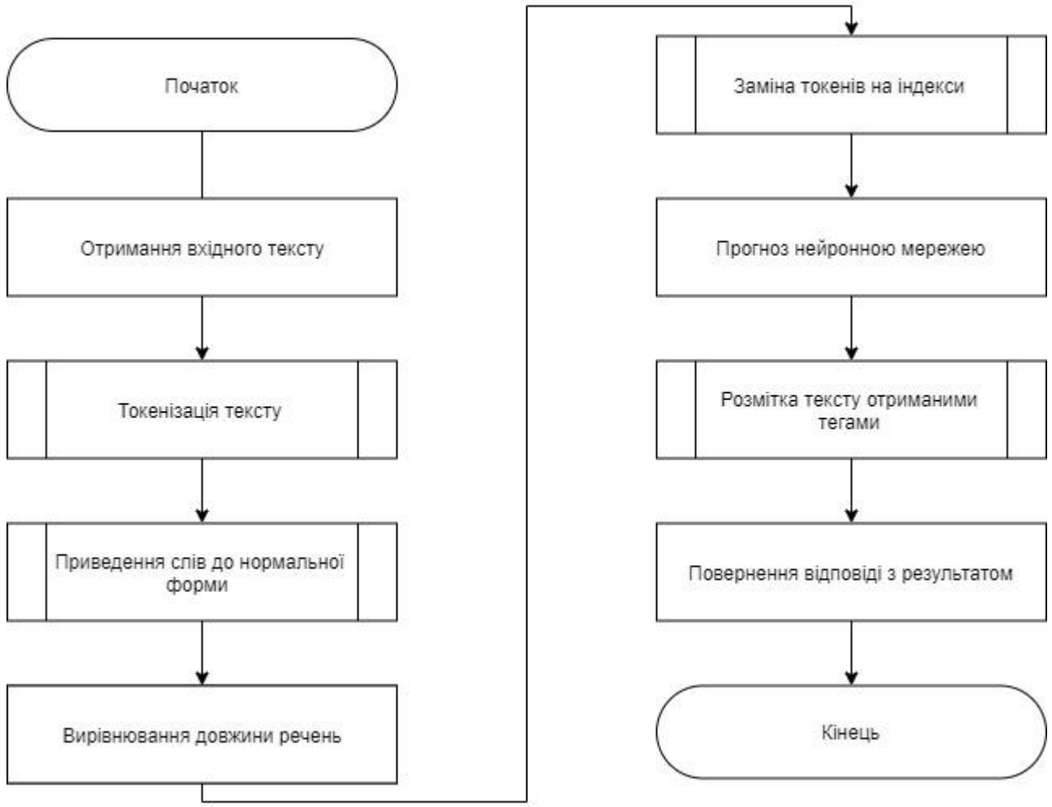
# ДОДАТОК А




| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | ДП 6128. 03.000 Д1 | |
| Зм. | Арк. | № докум. | Підпис | Дата | Розподілена система аналітичного опрацювання даних. Схема структурна | Літера / Маса / Масштаб |
| Розроб. | | Харчук В.В. | | | | |
| Перевір. | | Болдак А.О. | | | | Аркуш 1 / Аркушів 1 |
| Т. контр. | | | | | | |
| Н. контр. | | Сімоненко В.П. | | | | НТУУ "КПІ" ФІОТ Група ІО-61 |
| Затв. | | | | | | |




| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | ДП 6128. 04.000 Д2 | |
| | | | | | Інтеграція сервісу<br>Схема функціональна | Літера / Маса / Масштаб |
| Зм. | Арк. | № докум. | Підпис | Дата | | |
| Розроб. | | Харчук В.В. | | | | Аркуш 1 / Аркушів 1 |
| Перевір. | | Болдак А.О. | | | | |
| Т. контр. | | | | | | НТУУ "КПІ" ФІОТ<br>Група ІО-61 |
| Н. контр. | | Сімоненко В.П. | | | | |
| Затв. | | | | | | |




| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | ДП 6128. 05.000 ДЗ | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| Зм. | Арк. | № докум. | Підпис | Дата | Схема операцій над вхідними даними<br><br>Схема принципова | Літера / Маса / Масштаб |
| Розроб. | | Харчук В.В. | | | | |
| Перевір. | | Болдак А.О. | | | | Аркуш 1 / Аркушів 1 |
| Т. контр. | | | | | | |
| | | | | | | НТУУ “КПІ” ФІОТ<br>Група ІО-61 |
| Н. контр. | | Сімоненко В.Л. | | | | |
| Затв. | | | | | | |

# ДОДАТОК Б

```
import pandas as pd
import numpy as np
import tokenize_uk
import pickle
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from tensorflow.python.keras.backend import set_session
import tensorflow as tf
from keras.models import Model, Input, load_model
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
from keras.initializers import Constant
from sklearn.model_selection import train_test_split
from seqeval.metrics import f1_score, classification_report
from ner_nlp_extracting_service import ner_nlp_extracting
from pymongo import MongoClient
from vesum_db import vesum_service
import tensorflowjs as tfjs
import matplotlib.pyplot as plt
plt.style.use("ggplot")


def process_annotation(annotation):
types_dict = {
'ПЕРС': 'PERS',
'ЛОК': 'LOC',
'ОРГ': 'ORG',
'РІЗН': 'MISC',
'PERS': 'PERS',
```

```
'LOC': 'LOC',
'ORG': 'ORG',
'MISC': 'MISC'
}
ann_list = annotation.split('\t')
type_annotation = types_dict[ann_list[1].split(' ')[0]]
tokens = tokenize_uk.tokenize_words(ann_list[2])
return list(map(lambda x: (x, f'I-{type_annotation}') if x != tokens[0] else (x, f'B-{type_annotation}'), tokens))

DATA_DIR = 'data_raw/'

ready_data = list()

for dir_ in set(map(lambda x: x[:-4], listdir(DATA_DIR))):
if dir_ + '.txt' in listdir(DATA_DIR) and dir_ + '.ann' in listdir(DATA_DIR):
with open(DATA_DIR + dir_ + '.txt', 'r') as f:
text = f.read()
with open(DATA_DIR + dir_ + '.ann', 'r') as f:
annotations = f.read()

print(DATA_DIR + dir_ + '.ann')

tokens = list()
list(map(lambda x: tokens.extend(process_annotation(x)) if x else x, annotations.split('\n')))

text = map(lambda sentence: tokenize_uk.tokenize_words(sentence),
```

```
tokenize_uk.tokenize_sents(' '.join(tokenize_uk.tokenize_words(text))))

for sentence in text:
tmp_sentence = list()
for word in sentence:
if tokens and word == tokens[0][0]:
tmp_sentence.append(tokens.pop(0))
else:
tmp_sentence.append((word, 'O'))
ready_data.append(tmp_sentence)

with open("data_prepared/labeled_sentences.pkl", "wb") as f:
pickle.dump(ready_data, f)

vesum_service.init_vesum()
with open('data_prepared/labeled_sentences.pkl', 'rb') as f:
    data = pickle.load(f)
class Vesum_interface():
    MONGO_URL = ('VESUM_MONGO_DB_URL' in os.environ and os.environ['VESUM_MONGO_DB_URL']) or 'localhost:27017'
    DB_NAME = ('VESUM_DB_NAME' in os.environ and os.environ['VESUM_DB_NAME']) or 'natasha-uk-database'
    client = MongoClient(MONGO_URL, maxPoolSize=20)

    def get_main_form_from_vesum(self, word):
        samples = [doc['mainForm'] for doc in (self.client[self.DB_NAME]['vesum-entry'].find({'word': word}))]
        if len(set(samples)) == 1:
            return samples[0]
```

```
        else:
            return word
vesum = Vesum_interface()
words = list()
[[words.append(vesum.get_main_form_from_vesum(word[0])) for word in sentence] for sentence in tqdm(data)]
print(len(words))
words = list(set(words))
words.append("ENDPAD")
words.append("UNKNOWN")

word_embedding = dict() with open('fiction.cased.tokenized.word2vec.300d', 'r') as f: next(f) for line in tqdm(f.readlines()): word, value = line.split()[0], np.asarray(line.split()[1:], dtype='float32') word_embedding[word] = value
tags = list()

[[tags.append(word[1]) for word in sentence] for sentence in data]

print(len(tags))
tags = list(set(tags))
print(len(tags))

plt.hist([len(s) for s in data], bins=10)
plt.show()
word2indx = {w: i for i, w in enumerate(words)}
tag2indx = {t: i for i, t in enumerate(tags)}

embedding_matrix = np.zeros((len(word2indx), 300))
```

```
for word, i in tqdm(word2indx.items()):
    embedding_vector = word_embedding.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector


X = [[word2indx[vesum.get_main_form_from_vesum(w[0])] for w in s] for s in tqdm(data)]
X = pad_sequences(X, maxlen=70, padding='post', truncating='post', value=word2indx['ENDPAD'])


y = [[tag2indx[w[1]] for w in s] for s in data]
y = pad_sequences(y, maxlen=70, padding="post", truncating='post', value=tag2indx["O"])
y = [to_categorical(i, num_classes=len(tag2indx)) for i in y]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)


input = Input(shape=(70, ))
model = Embedding(input_dim=len(word2indx), output_dim=50, input_length=70,
                  #embeddings_initializer=Constant(embedding_matrix),
                  #trainable=False
                 )(input)
model = Dropout(0.1)(model)
model = Bidirectional(LSTM(units=32, return_sequences=True, recurrent_dropout=0.1))(model)
out = TimeDistributed(Dense(len(tag2indx), activation="softmax"))(model) # softmax output layer


model = Model(input, out)
```

```
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

model.summary()

print(f'Train f-score: {f1_score(y_true=[[tags[w] for w in sent] for sent in np.argmax(y_train, axis=-1)], y_pred=[[tags[w] for w in sent] for sent in np.argmax(model.predict(X_train, verbose=1), axis=-1)])}')

print(f'Test f-score: {f1_score(y_true=[[tags[w] for w in sent] for sent in np.argmax(y_test, axis=-1)], y_pred=[[tags[w] for w in sent] for sent in np.argmax(model.predict(X_test, verbose=1), axis=-1)])}')

print(classification_report(y_true=[[tags[w] for w in sent] for sent in np.argmax(y_test, axis=-1)], y_pred=[[tags[w] for w in sent] for sent in np.argmax(model.predict(X_test, verbose=1), axis=-1)]))

model.save('saved_objects/model_without_pretrained_embedding.h5')

with open('saved_objects/word2indx', 'wb') as f:
    pickle.dump(word2indx, f)

with open('saved_objects/tag2indx', 'wb') as f:
    pickle.dump(tag2indx, f)

vesum_service.init_vesum()

sess = tf.Session()

graph = tf.get_default_graph()

set_session(sess)

model = load_model('saved_objects/model_without_pretrained_embedding.h5')

vesum = Vesum_interface()

with open('saved_objects/word2indx', 'rb') as f:
    word2indx = pickle.load(f)

with open('saved_objects/tag2indx', 'rb') as f:
    tag2indx = pickle.load(f)

ner_nlp_extracting('біографія Івана Кузьменка досить цікава',
                   model, vesum, word2indx, tag2indx, sess, graph)
```